Image Processing and Filtering

Robert Haase

With material from

Marcelo Leomil Zoccoler and Till Korten, PoL TU Dresden

Mauricio Rocha Martins, Norden lab, MPI CBG

Dominic Waithe, Oxford University

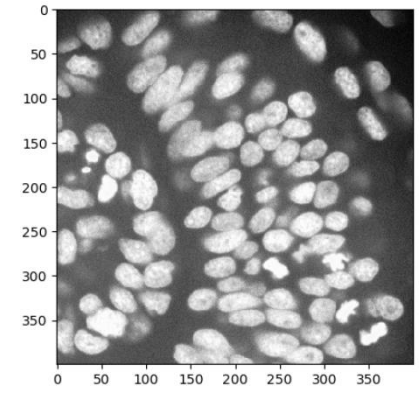
Alex Bird, Dan White, MPI CBG

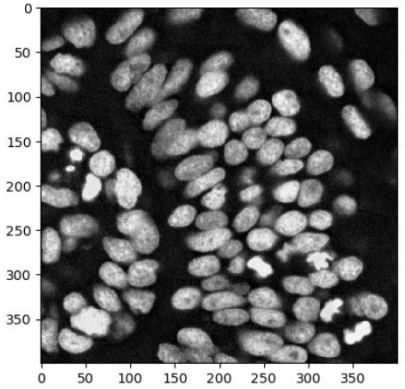
April 2023

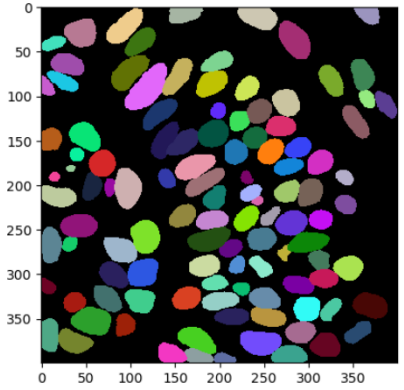
@haesleinhuepf

Lecture overview: Bio-image Analysis

- Image Data Analysis workflows
- Goal: **Quantify observations, substantiate conclusions with numbers**

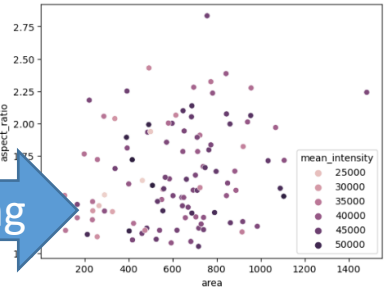


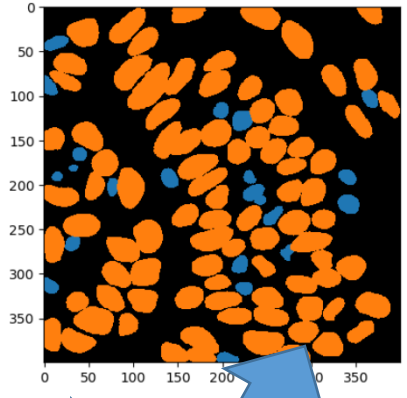




	area	mean_intensity	major_axis_length	minor_axis_length
0	594.0	40572.809764	28.611591	26.537947
1	645.0	43764.872868	33.511511	24.566916
2	1105.0	51970.561991	45.232031	31.456308
3	718.0	47015.487465	31.023274	29.5
4	791.0	49132.515803	36.382253	27.7
...	...	...	...	
105	238.0	30477.126050	20.252197	15.2
106	615.0	32886.154472	41.030760	19.8
107	110.0	33042.445455	14.366347	9.9
108	222.0	43304.180180	25.370549	11.6
109	167.0	43378.808383		

Plotting





Visualization

Image filtering → Image segmentation → Feature extraction → Object classification

Bio-image analysis → Bio-statistics

PoL
Physics of Life
TU Dresden

CENTER FOR
SYSTEMS BIOLOGY
DRESDEN

Image Visualization

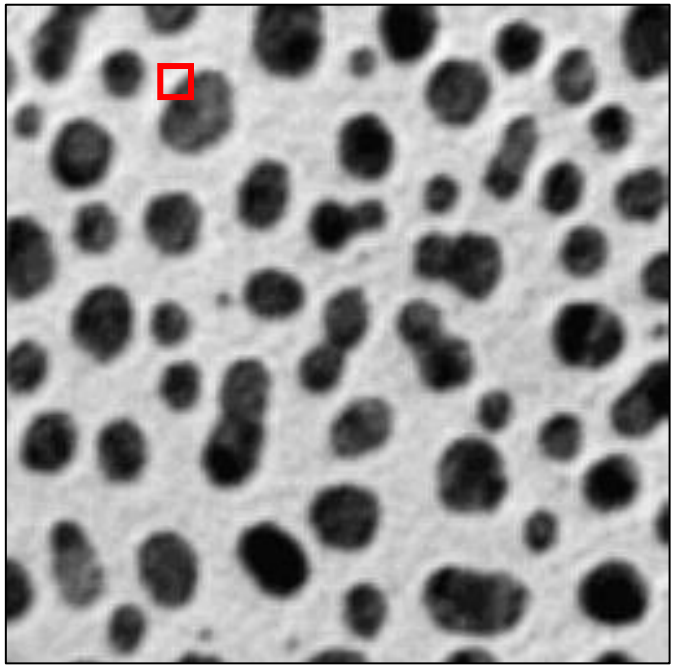
Robert Haase

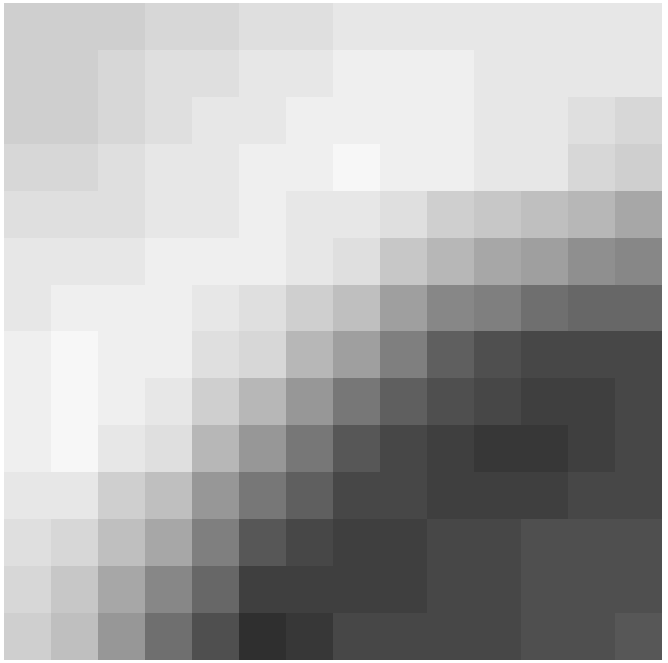
April 2023

@haesleinhuepf

Images and pixels

- An image is just a matrix of numbers
- Pixel: “picture element”
- The edges between pixels are an artefact of the imaging / digitization. They are not real!

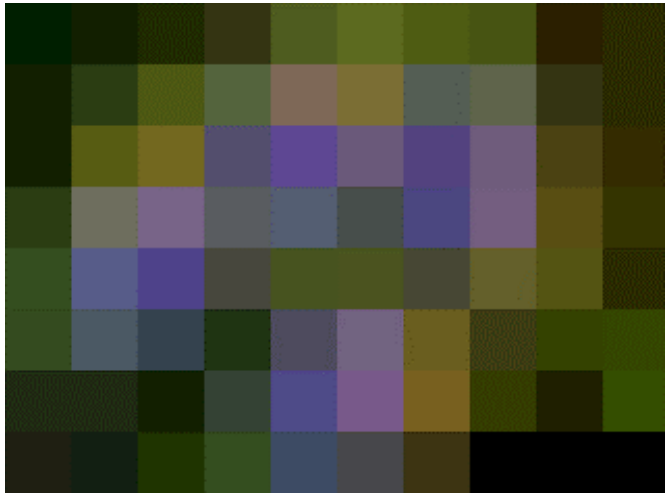




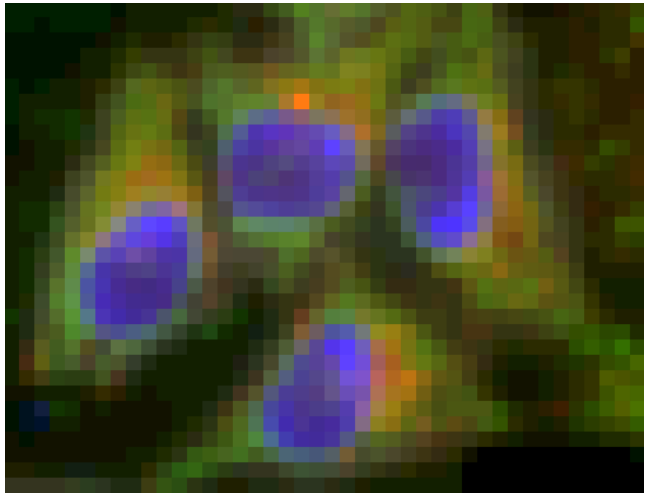
48	48	48	40	40	32	32	24	24	24	24	24	24	24
48	48	40	32	32	24	24	16	16	16	24	24	24	24
48	48	40	32	24	24	16	16	16	16	24	24	32	40
40	40	32	24	24	16	16	8	16	16	24	24	40	48
32	32	32	24	24	16	24	24	32	48	56	64	72	88
24	24	24	16	16	16	24	32	56	72	88	96	112	120
24	16	16	16	24	32	48	64	96	120	128	144	152	152
16	8	16	16	32	40	72	96	128	160	176	184	184	184
16	8	16	24	48	72	104	136	160	176	184	192	192	184
16	8	24	32	72	104	136	168	184	192	200	200	192	184
24	24	48	64	104	136	160	184	184	192	192	192	184	184
32	40	64	88	128	168	184	192	192	184	184	176	176	176
40	56	88	120	152	192	192	192	192	184	184	176	176	176
48	64	104	144	176	208	200	184	184	184	184	176	176	168

Pixel size versus resolution

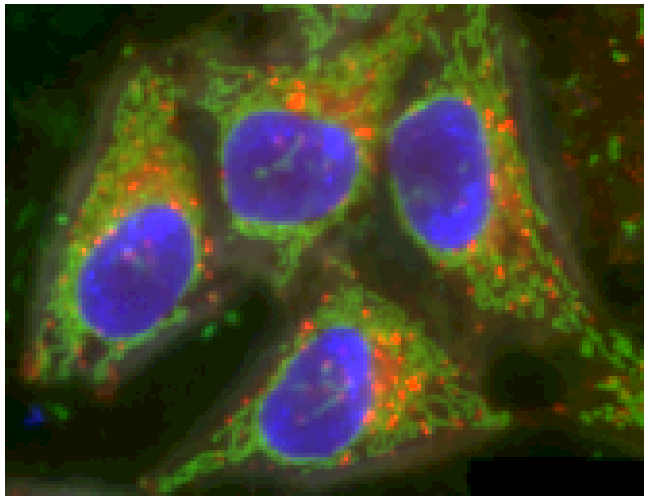
- Pixel size is a digital property of an image.
- You configure it during the imaging session at the microscope.



Pixel size: 3.3 µm



Pixel size: 0.8 µm

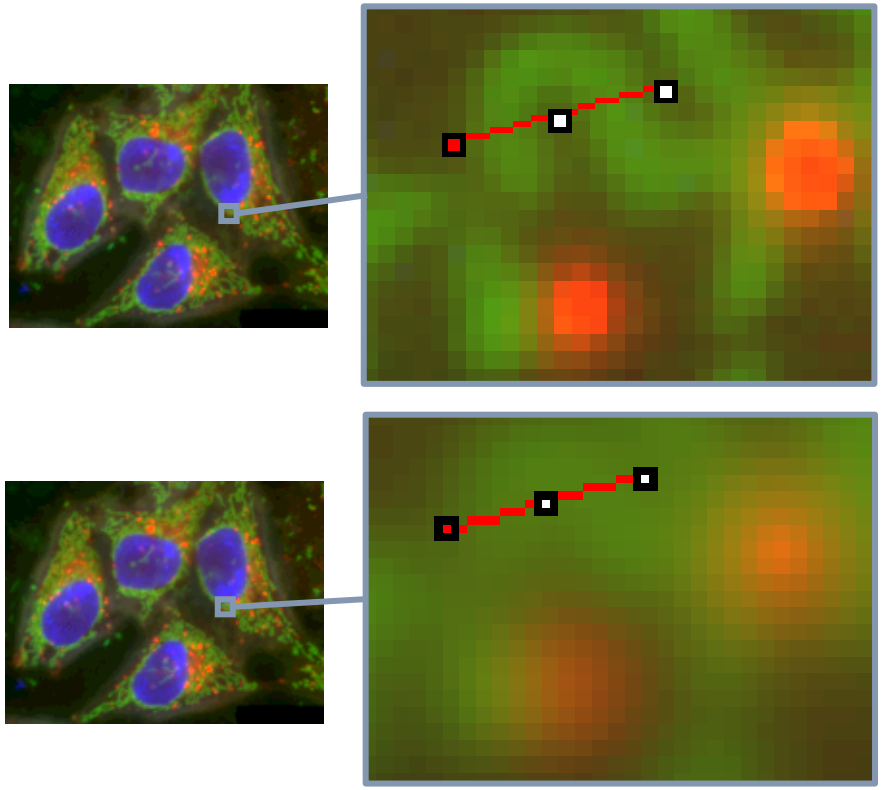


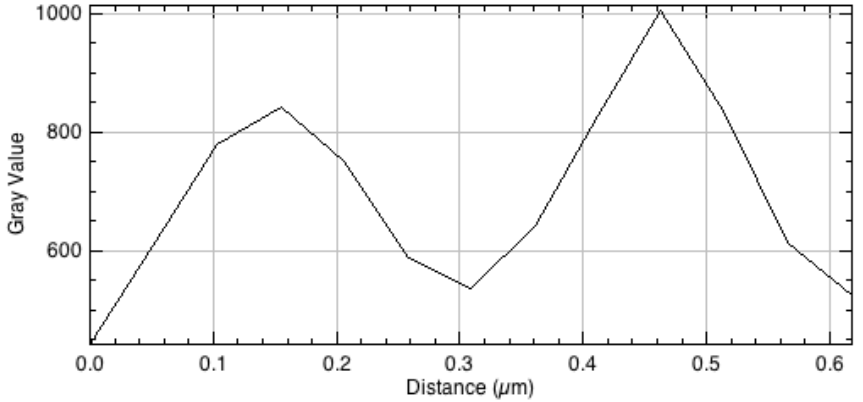
Pixel size: 0.05 µm

- We are not talking about resolution!

Pixel size versus resolution

- Resolution is a property of your imaging system.
- The measure of how close object can be in an image while still being differentiable, is called spatial resolution.





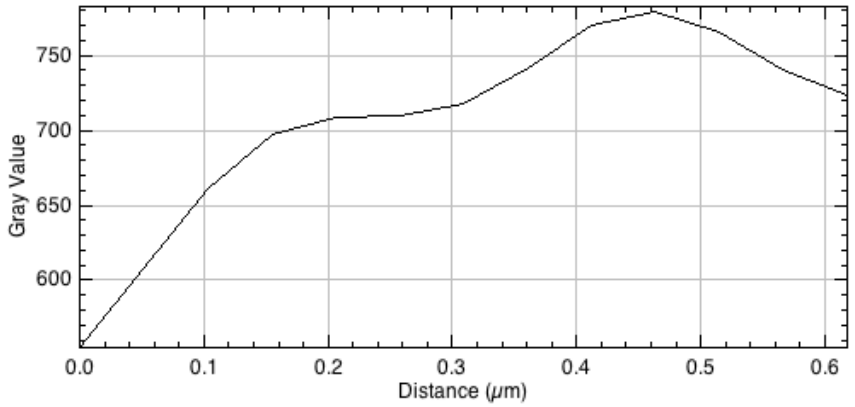
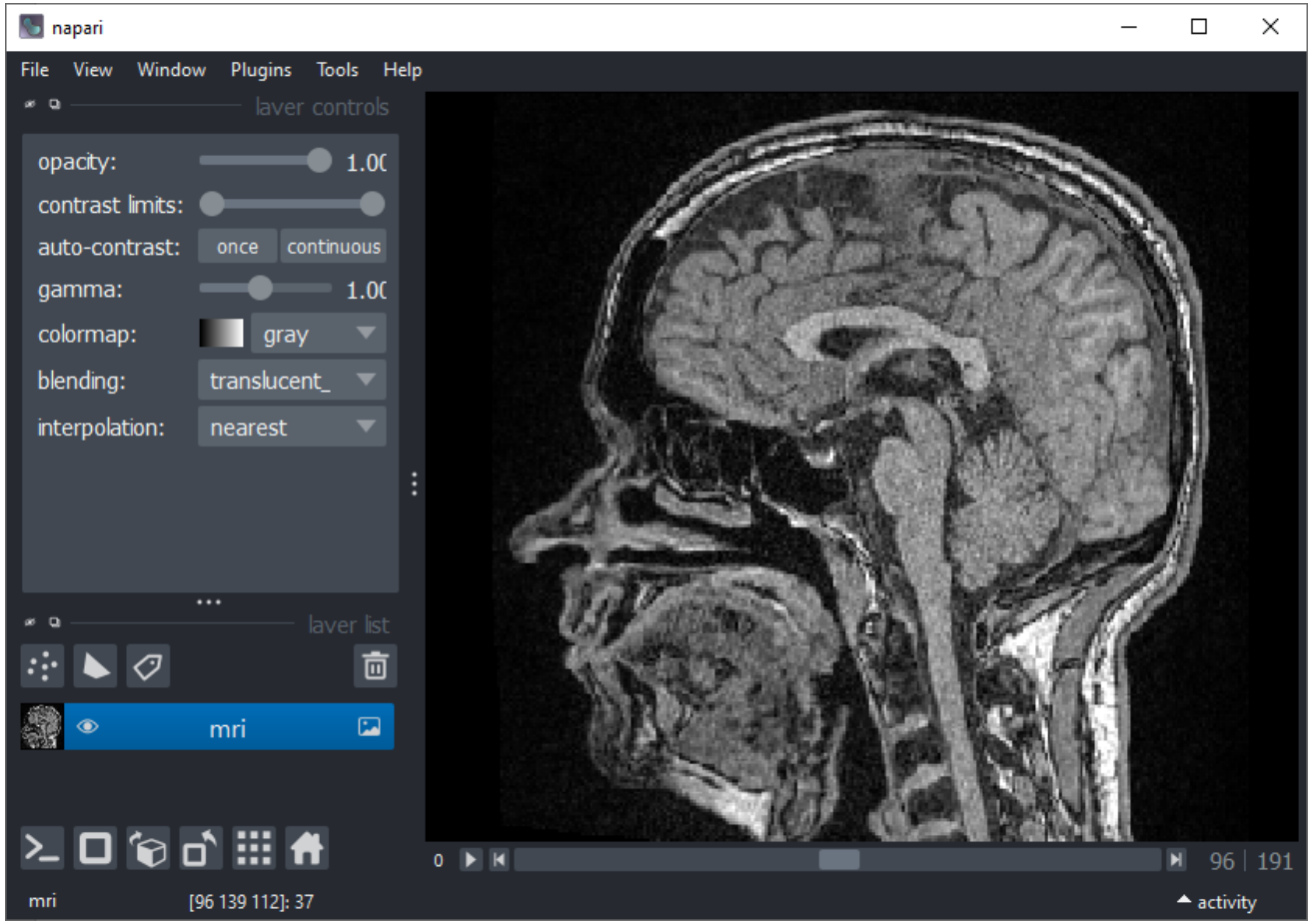


Image stacks and voxels

There are tools available for exploring them



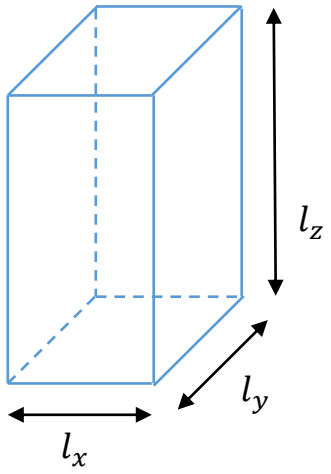

```
import stackview

stackview.slice(mri_image)
```

https://napari.org/

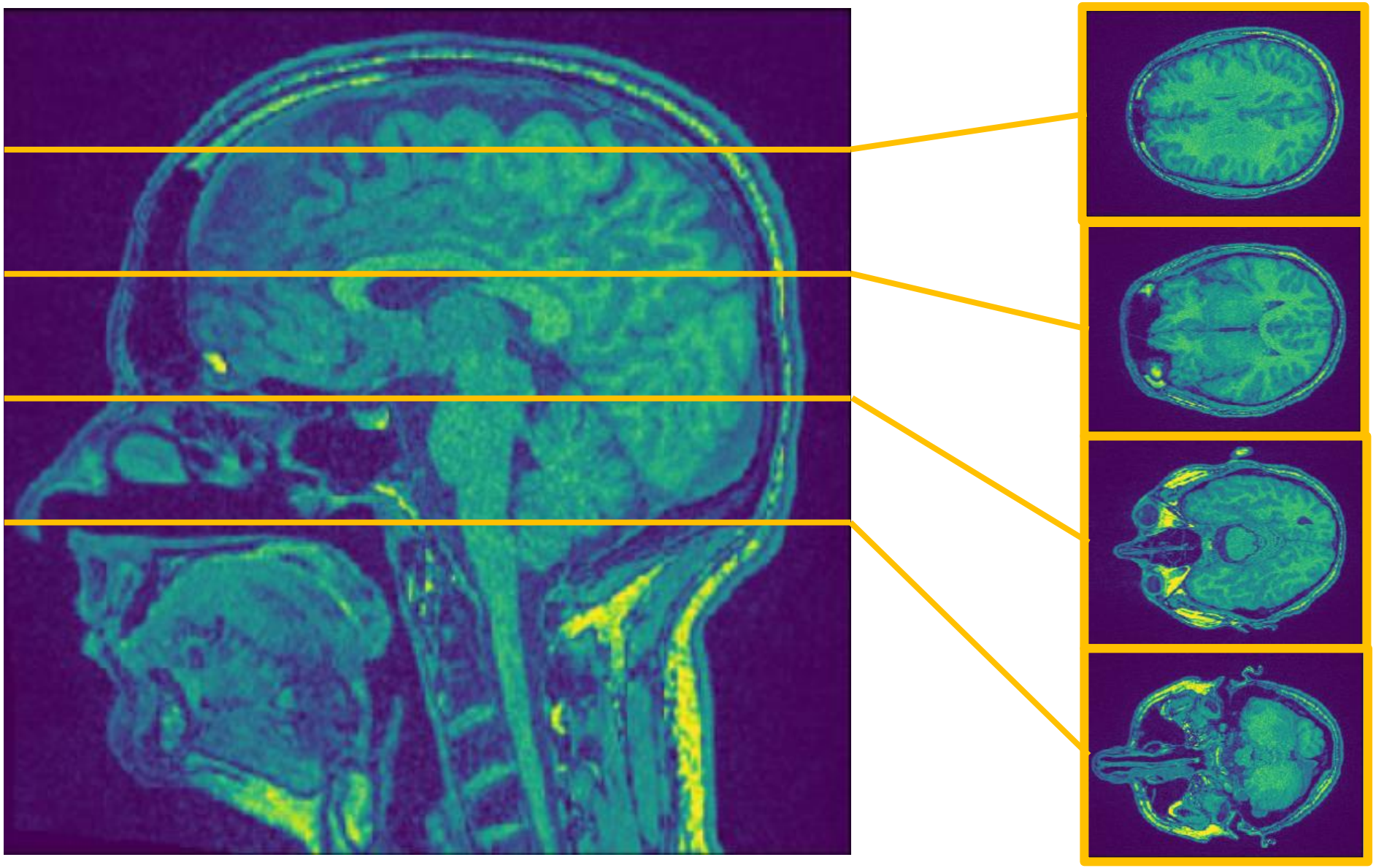
https://github.com/haesleinhuepf/stackview

Image stacks and voxels

- 3-dimensional images consisting of voxels
- “Image stack”
- Often anisotropic (not equally large in all directions)

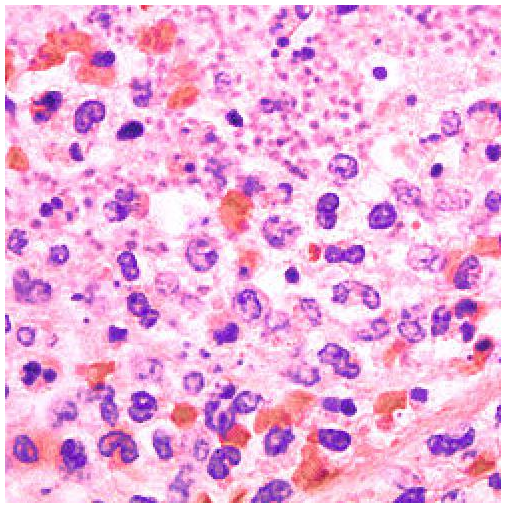


$$l_x = l_y \neq l_z$$



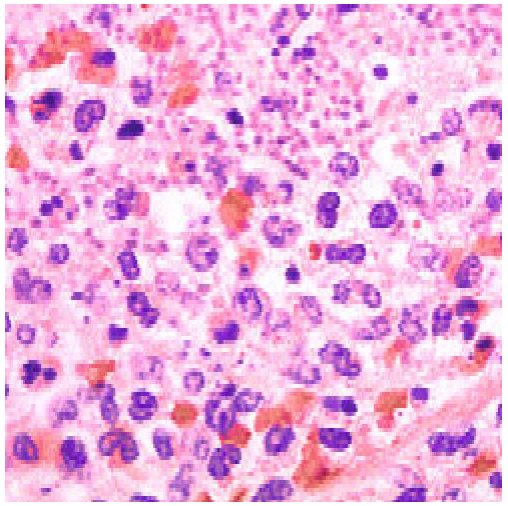
Anisotropy

- Voxel size has immediate impact on image quality and thus, on processing / analysis results.



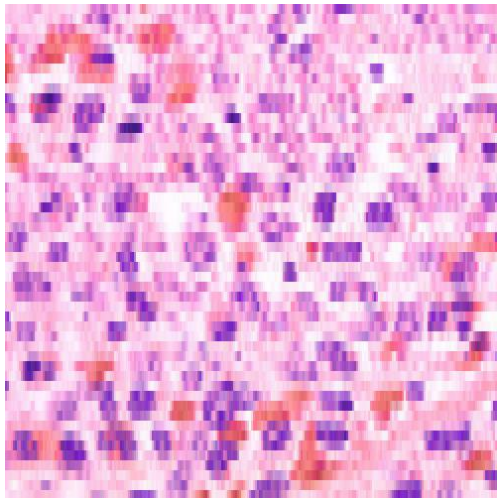
1:1

250 x 250 px



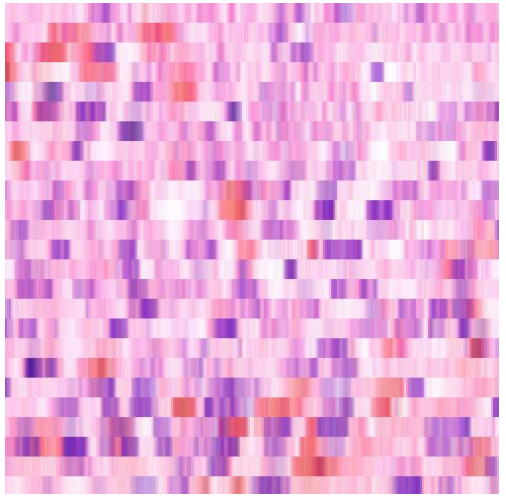
1:2

250 x 125 px



1:5

250 x 50 px



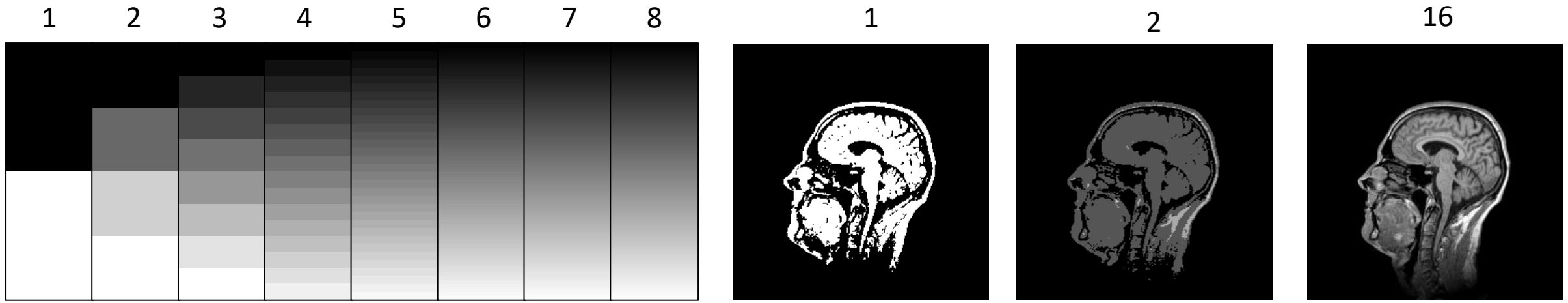
1:10

250 x 25 px

Image source: cropped from
https://de.m.wikipedia.org/wiki/Datei:Histo_Lungenpest.jpg

Bit-depth

- A bits is the smallest memory unit in computers, *atomic data*.
- The bit-depth *n* enumerates how many different intensity values are present in an image:
  - 2^n grey values
- In microscopy, images are usually stored as 8, 12 or 16-bit images.



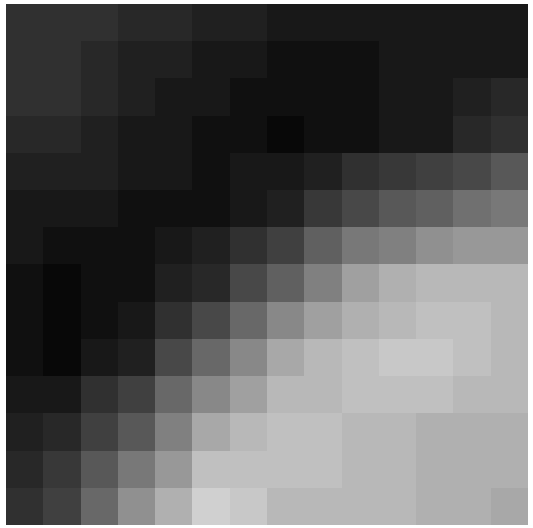
Colormaps / lookup tables

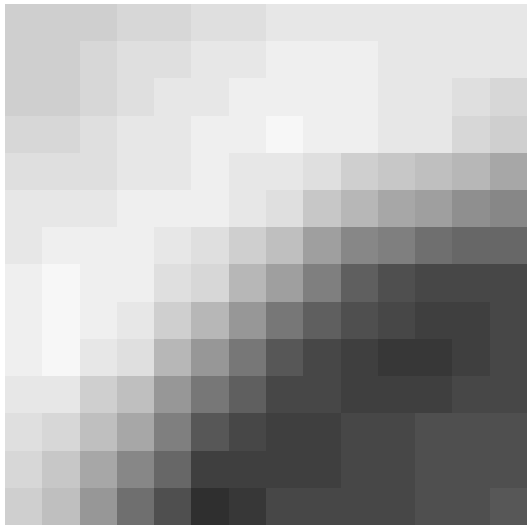
- The lookup table decides how the image is displayed on screen.
- Applying a different lookup table does not change the image. All pixel values stay the same, they just appear differently

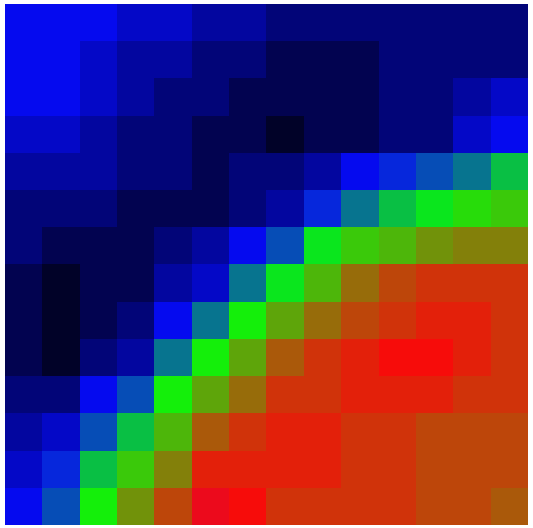
Pixel value	Display color
0	
1	
2	
…	
255	

Pixel value	Display color
0	
1	
2	
…	
255	

Pixel value	Display color
0	
1	
2	
…	
255	



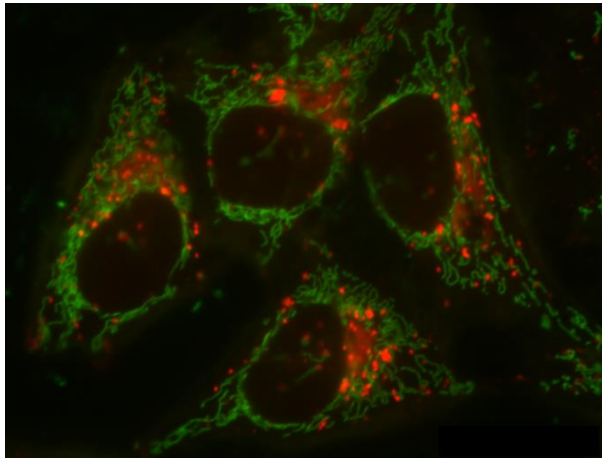




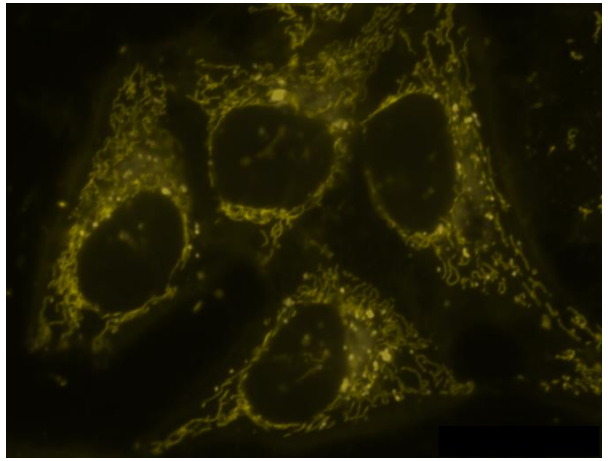
Colormaps / lookup tables

- Choose visualization of your color tables wisely!
- Think of people with red/green blindness!

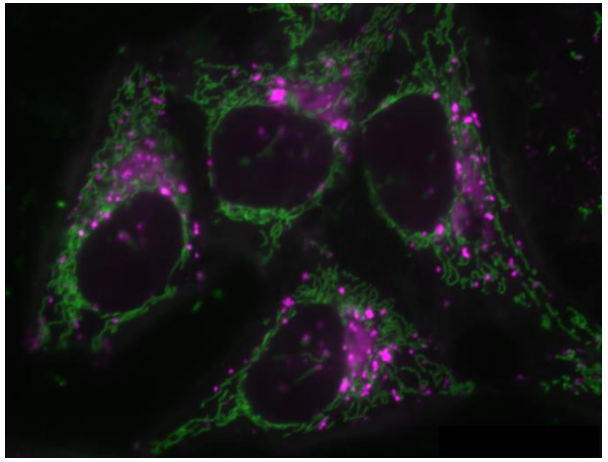
Common view

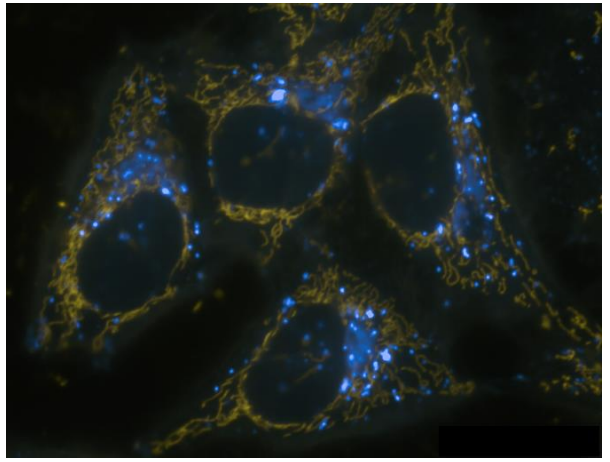


Red/green blind people may see it like



Replace red with magenta!

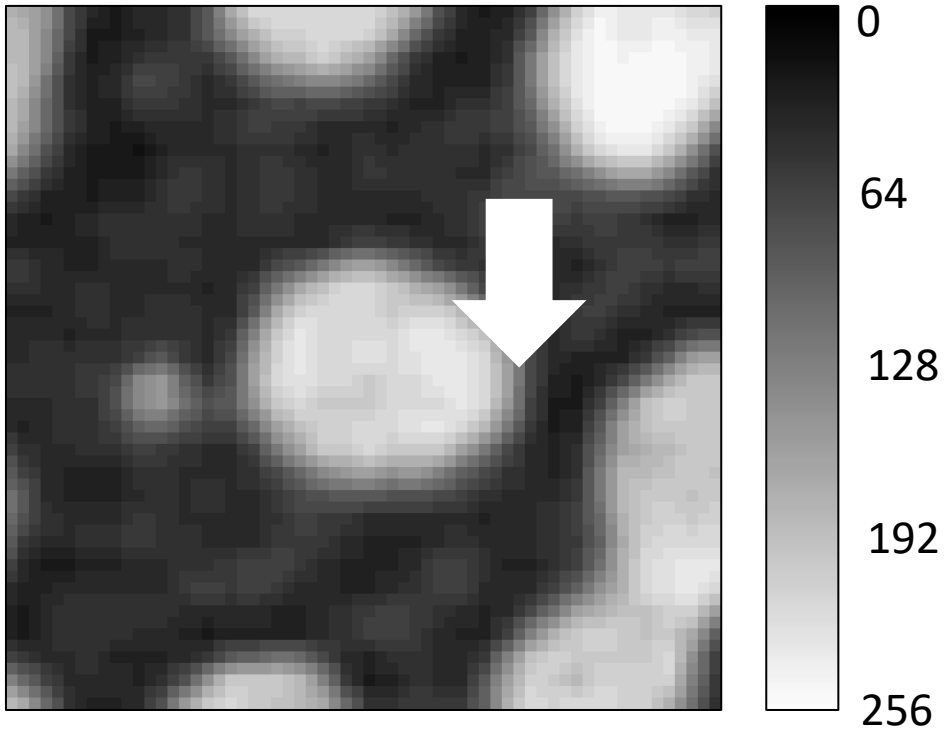


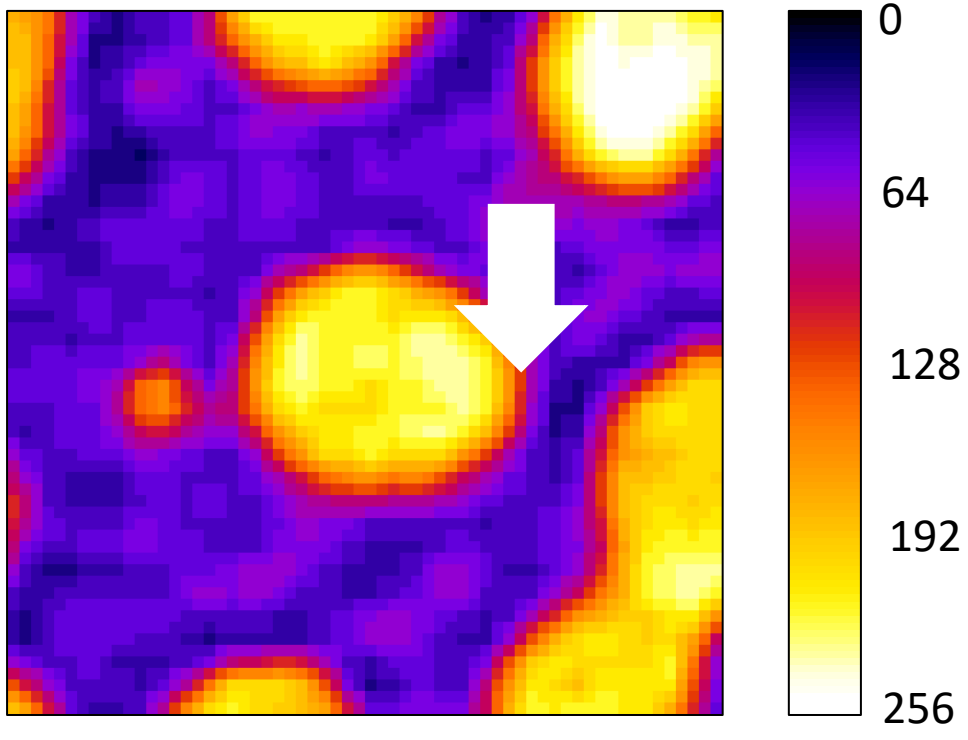


Colormaps / lookup tables

PoL Physics of Life TU Dresden

- Which intensity does the marked pixel have?

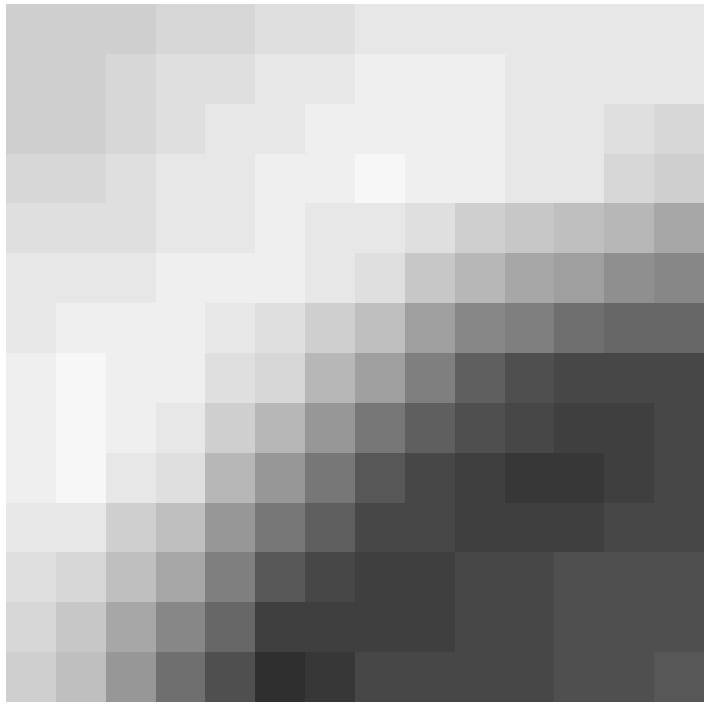


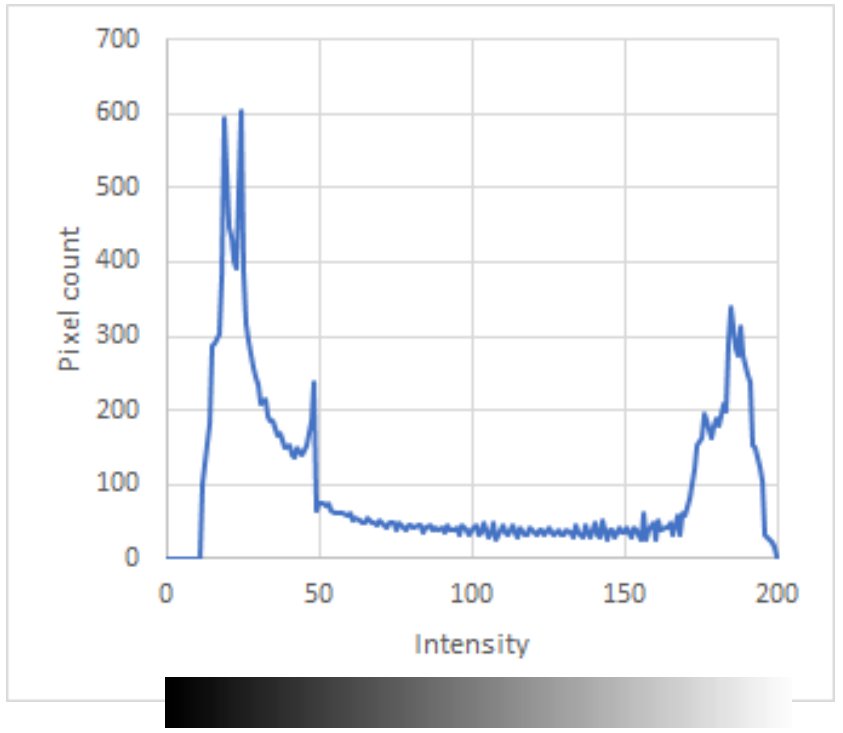


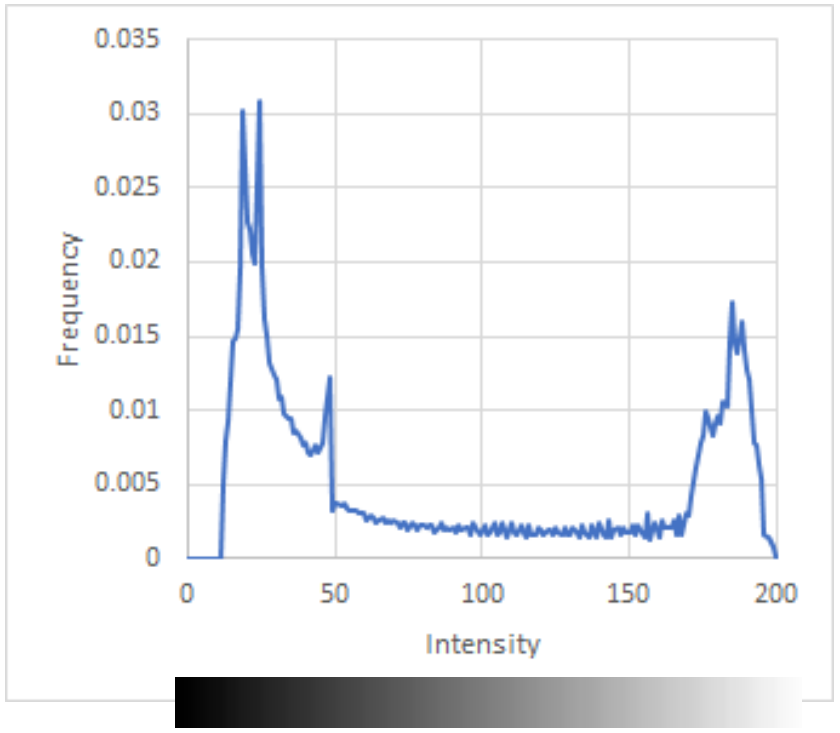
0	64	128	192

Histograms

- A histogram shows the probability distribution of pixel intensities.
- The probability of a pixel having a certain grey value can be measured by counting pixels and calculating the frequency of the given intensity.
- Whenever you see a histogram, try to imagine the lookup-table on the X-axis

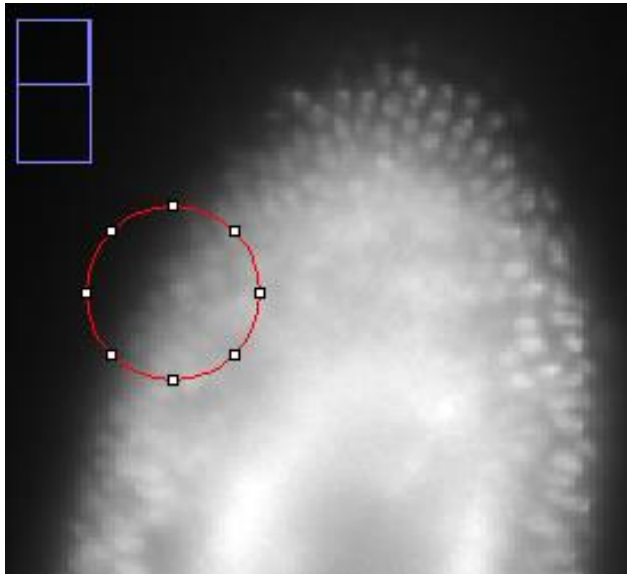


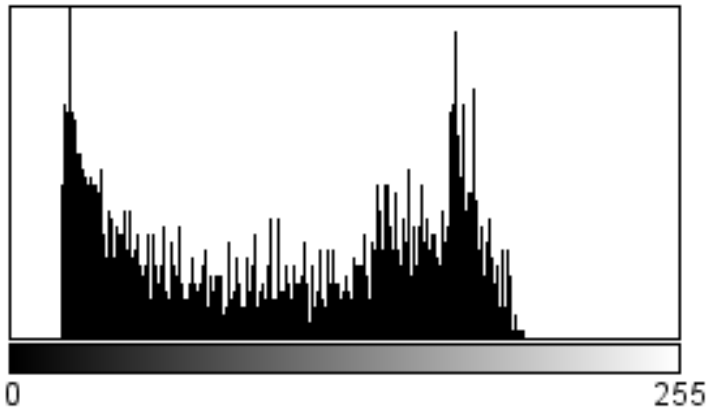




Histograms

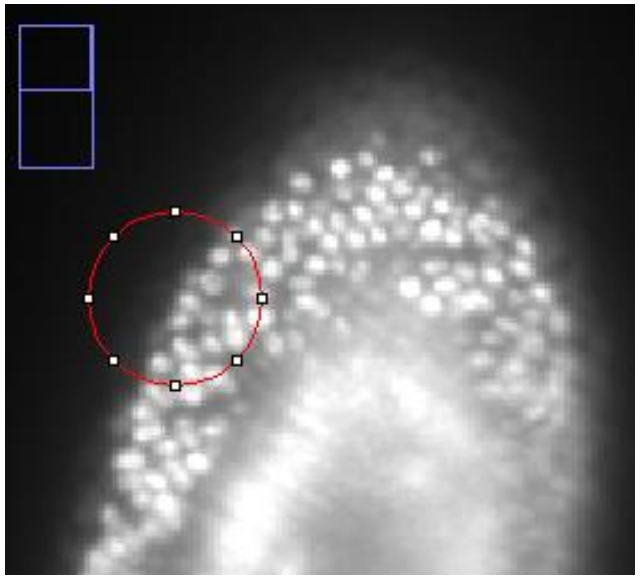
- Histograms are summaries of images
- Tell stories, e.g. about image quality

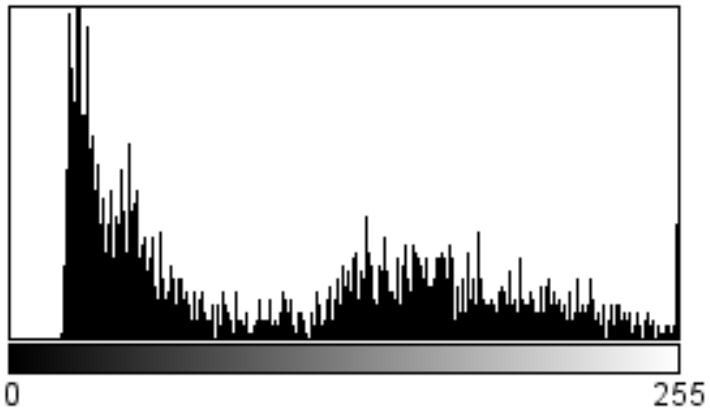




Count: 2053
Mean: 103.818
StdDev: 57.093

Min: 19
Max: 196
Mode: 22 (41)



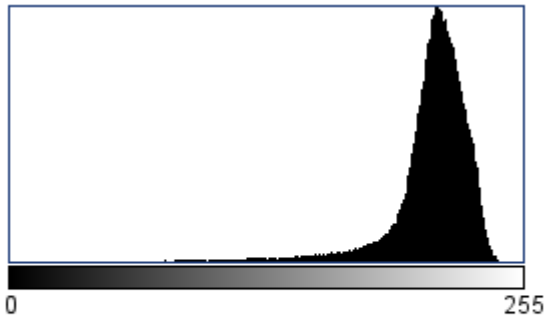


Count: 2053
Mean: 103.370
StdDev: 70.260

Min: 19
Max: 255
Mode: 25 (49)

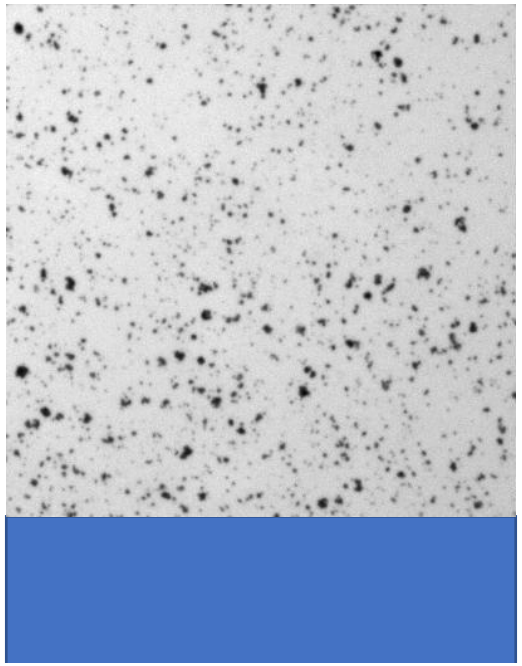
Histograms

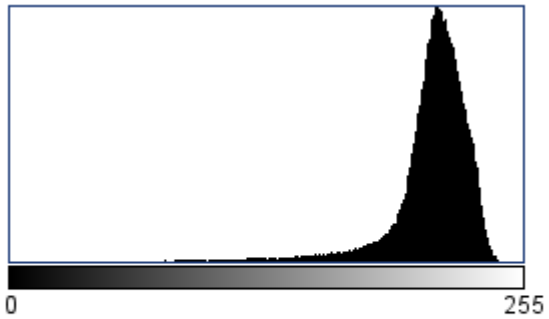
- To which of the three images does this histogram belong to?



N: 165648
Mean: 207.819
StdDev: 25.834
Value: 200

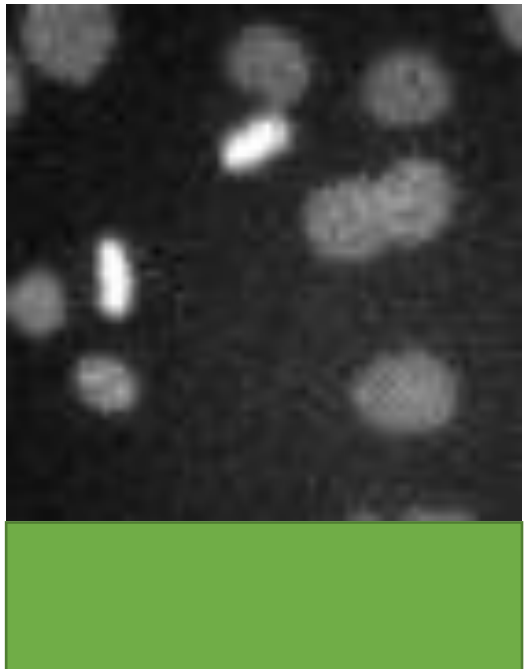
Min: 1
Max: 253
Mode: 212 (5234)
Count: 2219

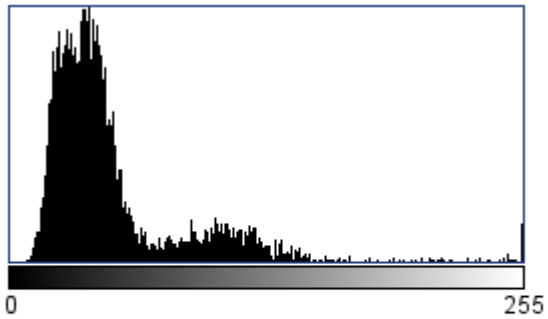




N: 165648
Mean: 207.819
StdDev: 25.834
Value: 200

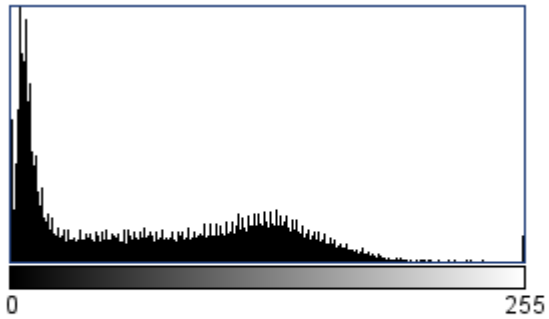
Min: 1
Max: 253
Mode: 212 (5234)
Count: 2219





N: 4900
Mean: 51.060
StdDev: 36.426
Value: 64

Min: 8
Max: 255
Mode: 39 (125)
Count: 9



N: 65536
Mean: 70.929
StdDev: 59.567
Value: 59

Min: 0
Max: 255
Mode: 4 (2352)
Count: 239

PoL Physics of Life TU Dresden

CENTER FOR SYSTEMS BIOLOGY DRESDEN

Image Filtering

Robert Haase

With material from

Marcelo Leomil Zoccoler and Till Korten, PoL, TU Dresden

April 2023

@haesleinhuepf

Filters

- An image processing filter is an operation on an image.
- It takes an image and produces a new image out of it.
- Filters change pixel values.

- There is no “best” filter. Which filter fits your needs, depends on the context.
- Filters do not do magic. They can not make things visible which are not in the image.

- Application examples
  - Noise-reduction
  - Artefact-removal
  - Contrast enhancement
  - Correct uneven illumination

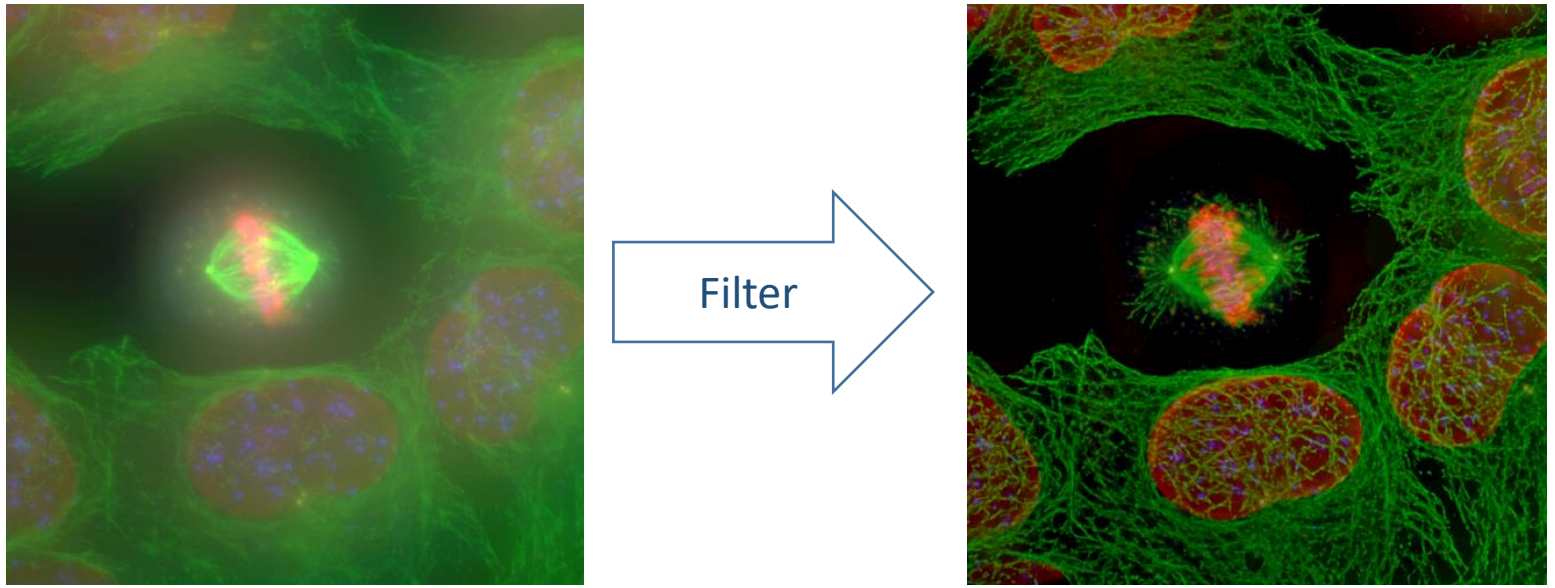
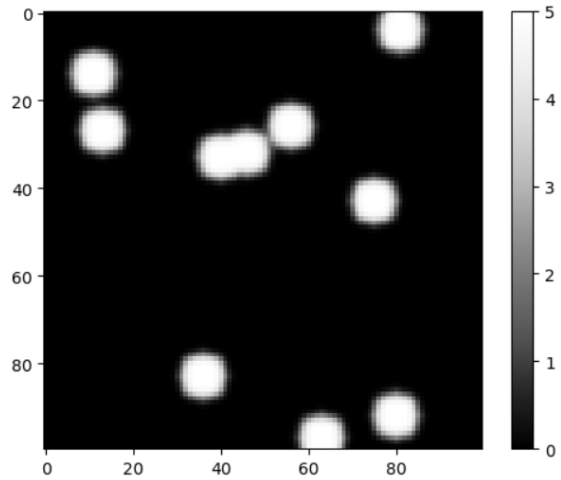


Image source: Alex Bird / Dan White MPI CBG

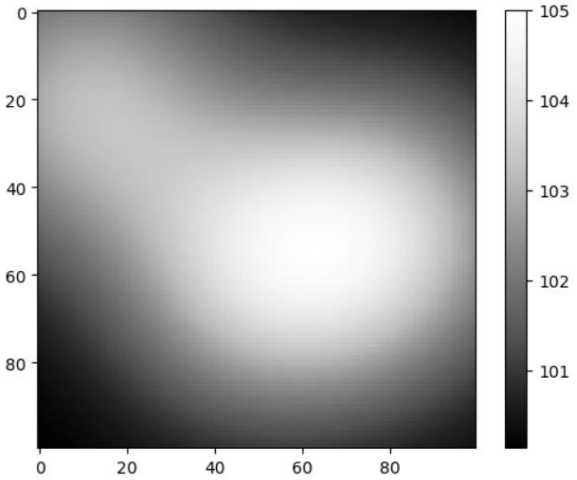
Effects harming image quality

- Image formation (simulated)

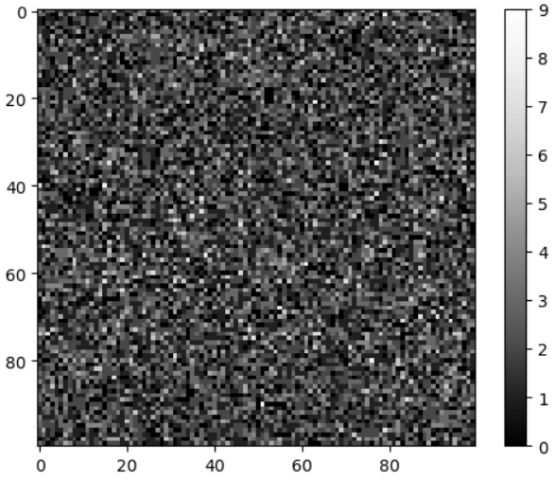
“nuclei”



“background”



“noise”



- Aberrations, defocus
- Motion blur

- Light from objects behind and in front of the scene (out-of-focus light)
- Dirt on the object slide
- Camera offset

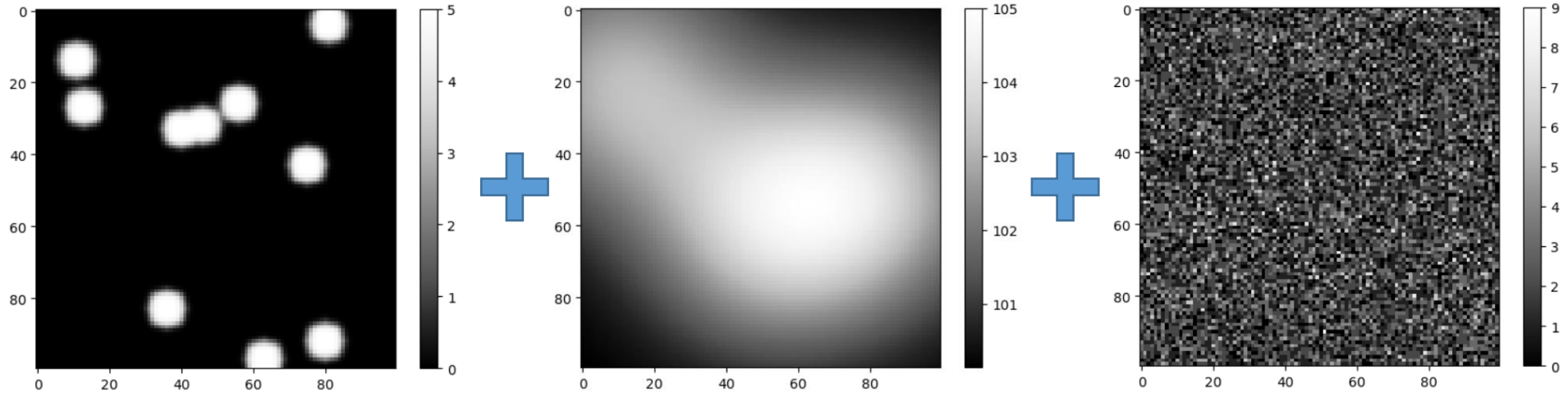
- Shot noise (arriving photons)
- Dark noise (electrons made from photons)
- Read-out-noise (electronics)

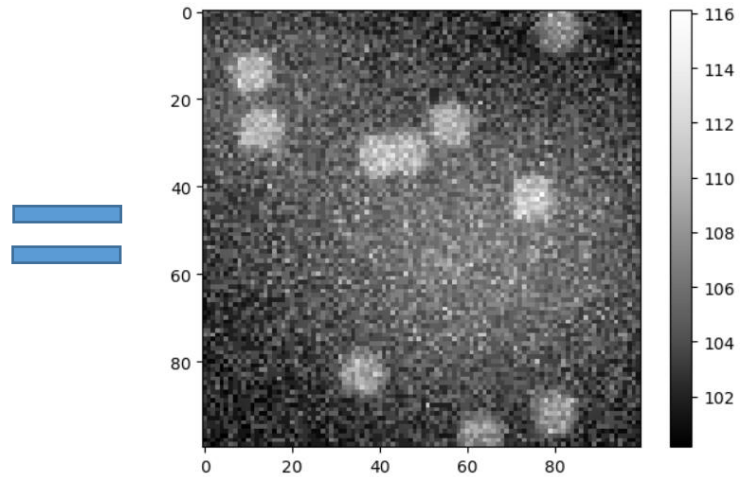
https://github.com/BiAPoL/Bio-Image_Analysis_with_Python/blob/49a787514a367829c3e0e1832f6cc533e96d549f/03_image_processing/simulated_dataset.ipynb

Effects harming image quality

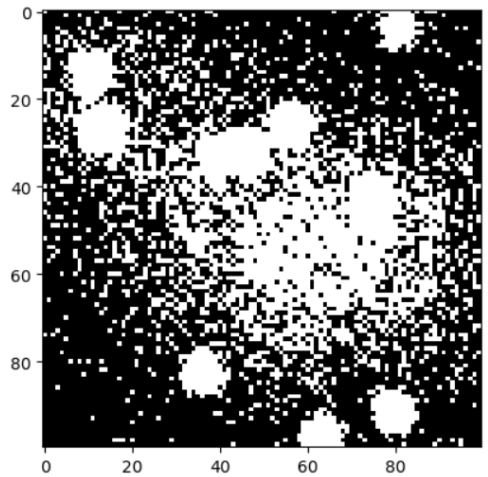
- Image formation (simulated)

“nuclei” “background” “noise”





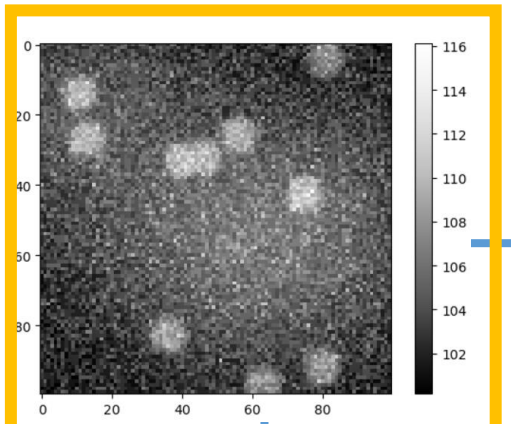
Segmentation



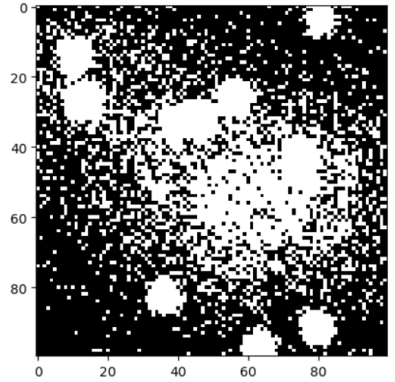
https://github.com/BiAPoL/Bio-Image_Analysis_with_Python/blob/49a787514a367829c3e0e1832f6cc533e96d549f/03_image_processing/simulated_dataset.ipynb

Image filtering

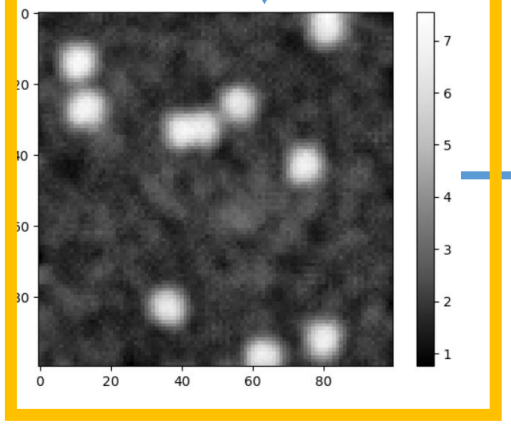
- We need to remove the noise to help the computer *interpreting* the image



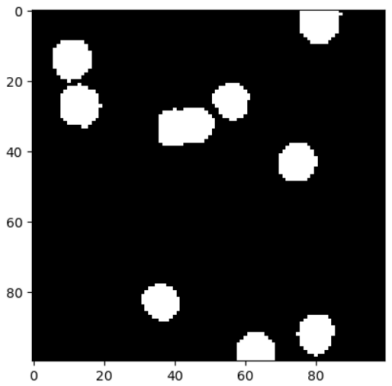
Segmentation



Filtering



Segmentation

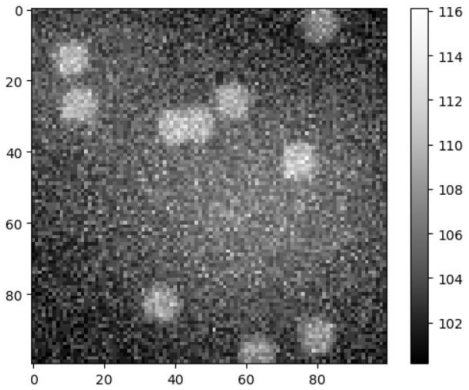


Oh no! I see thousands of tiny white objects!

Ok, it's just 9 objects.

Image filtering

- Attempt to invert / “undo” processes disturbing image quality



?

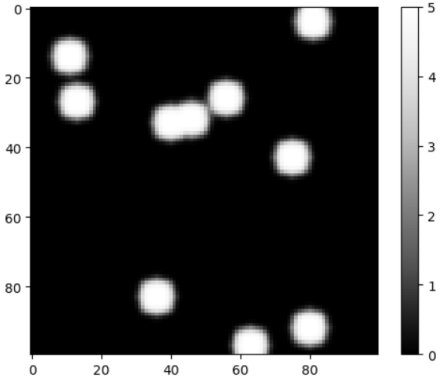
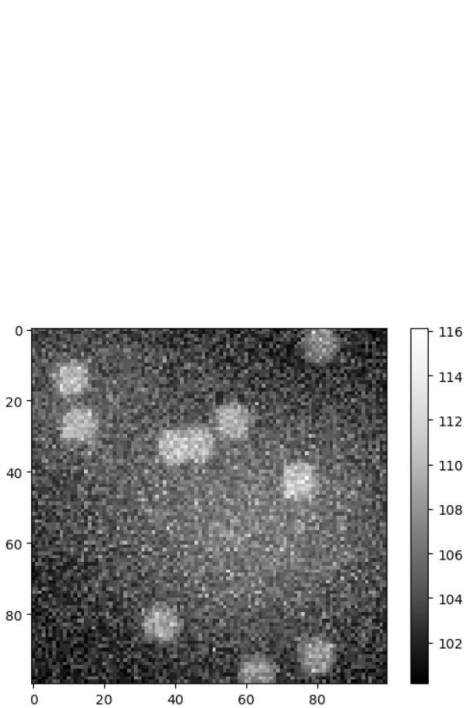
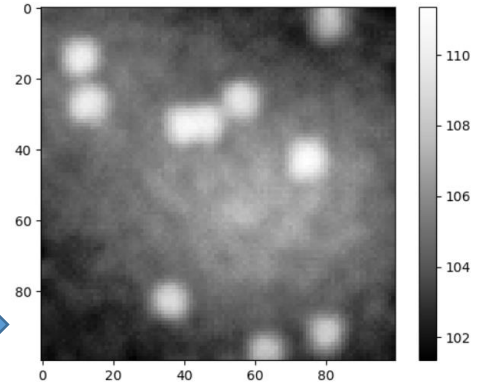


Image filtering

- Attempt to invert / “undo” processes disturbing image quality

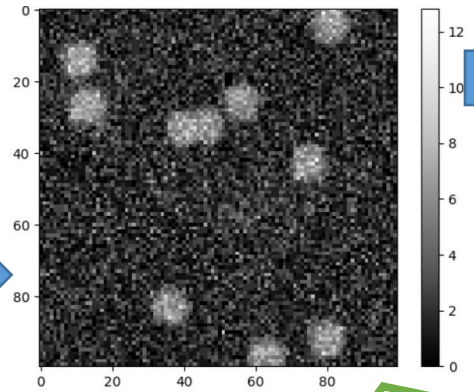


Denoise

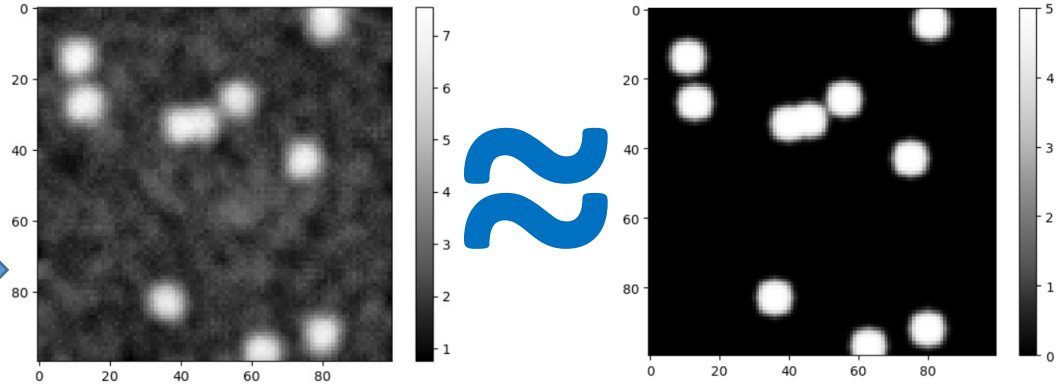


Low-pass filter:
Allows low frequencies pass

Remove background



Denoise



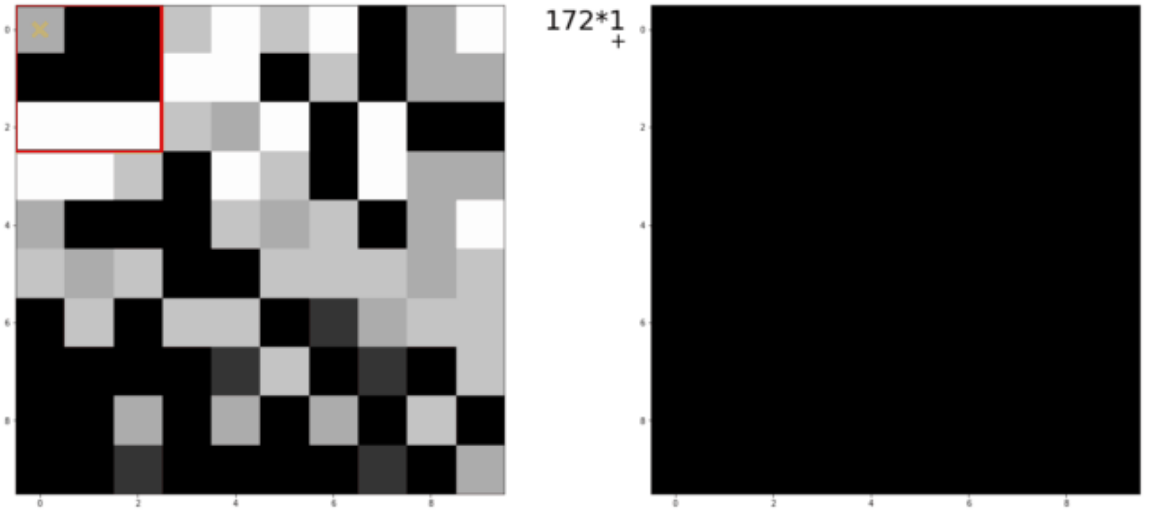
Band-pass filter:
Allows a specific range of frequencies pass

High-pass filter:
Allows high frequencies pass

https://github.com/BiAPoL/Bio-Image_Analysis_with_Python/blob/49a787514a367829c3e0e1832f6cc533e96d549f/03_image_processing/simulated_dataset.ipynb

Linear Filters

- *Linear filters* replace each pixel value with a weighted linear combination of surrounding pixels
- Filter *kernels* are matrices describing a linear filter
- This multiplication of surrounding pixels according to a matrix is called *convolution*



Mean filter, 3x3 kernel

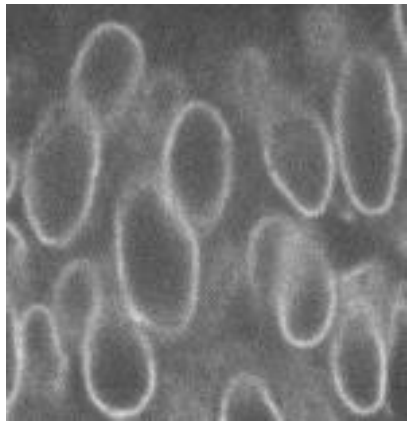
$$\begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Animation source: Dominic Waithe, Oxford University
https://github.com/dwaithe/generalMacros/tree/master/convolution_ani

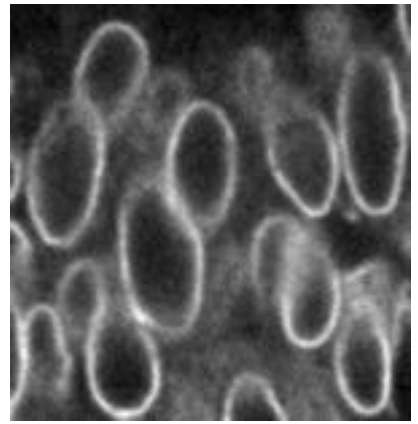
Linear filters

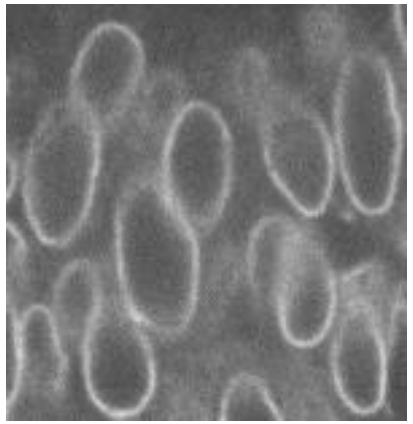
- Terminology:
  - “We convolve an image with a kernel.”
  - Convolution operator: *

- Examples
  - Mean
  - Gaussian blur
  - Sobel-operator
  - Laplace-filter

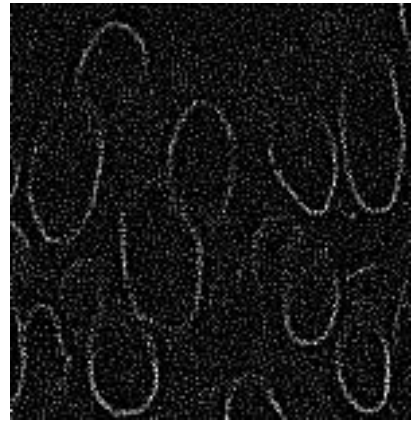


$$*\begin{bmatrix} 1 & 1 & 1 \\ 1 & 8 & 1 \\ 1 & 1 & 1 \end{bmatrix}=$$





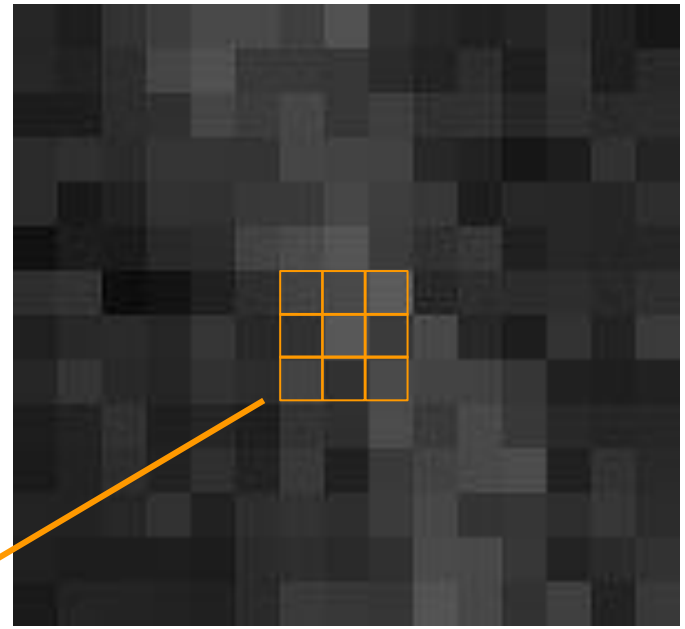
$$*\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}=$$



Nonlinear Filters

- Non linear filters also replace pixel value inside as rolling window but using a non-linear function.

- Examples: order statistics filters
  - Min
  - Median
  - Max
  - Variance
  - Standard deviation



75	85	60
67	73	91
50	88	59

50 59 60 67 73 75 85 88 91

Min Median Max

Noise removal

- Gaussian filter
- Median filter (computationally expensive)

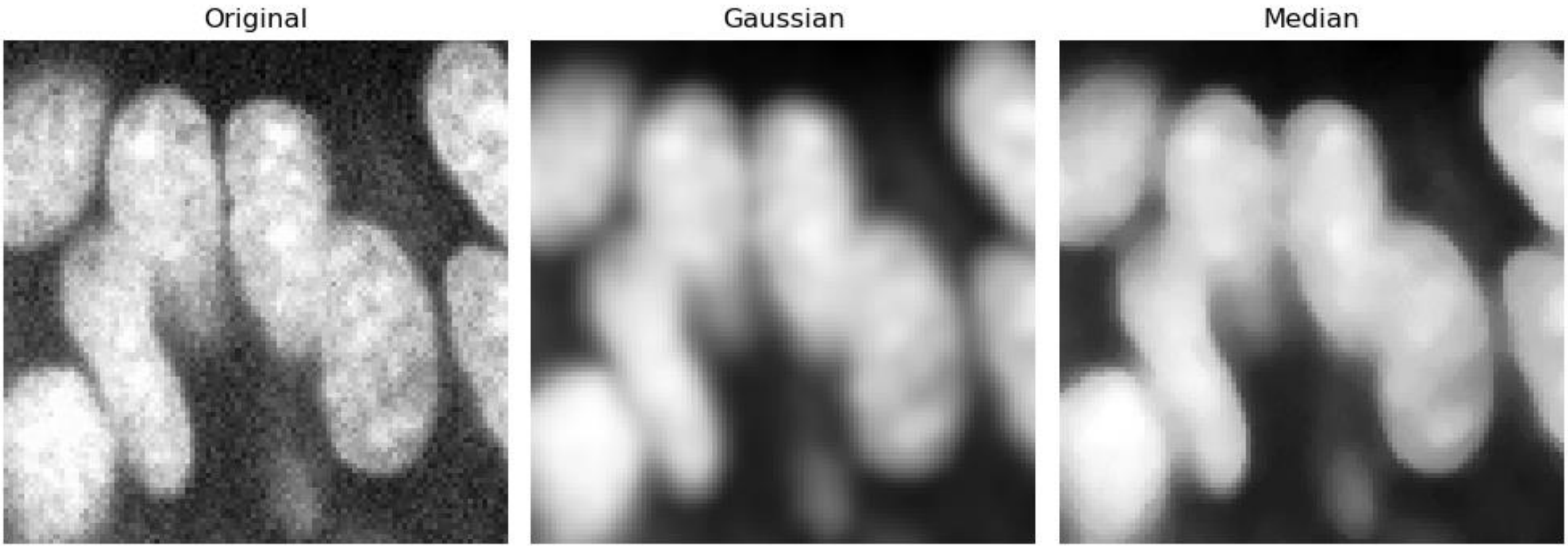
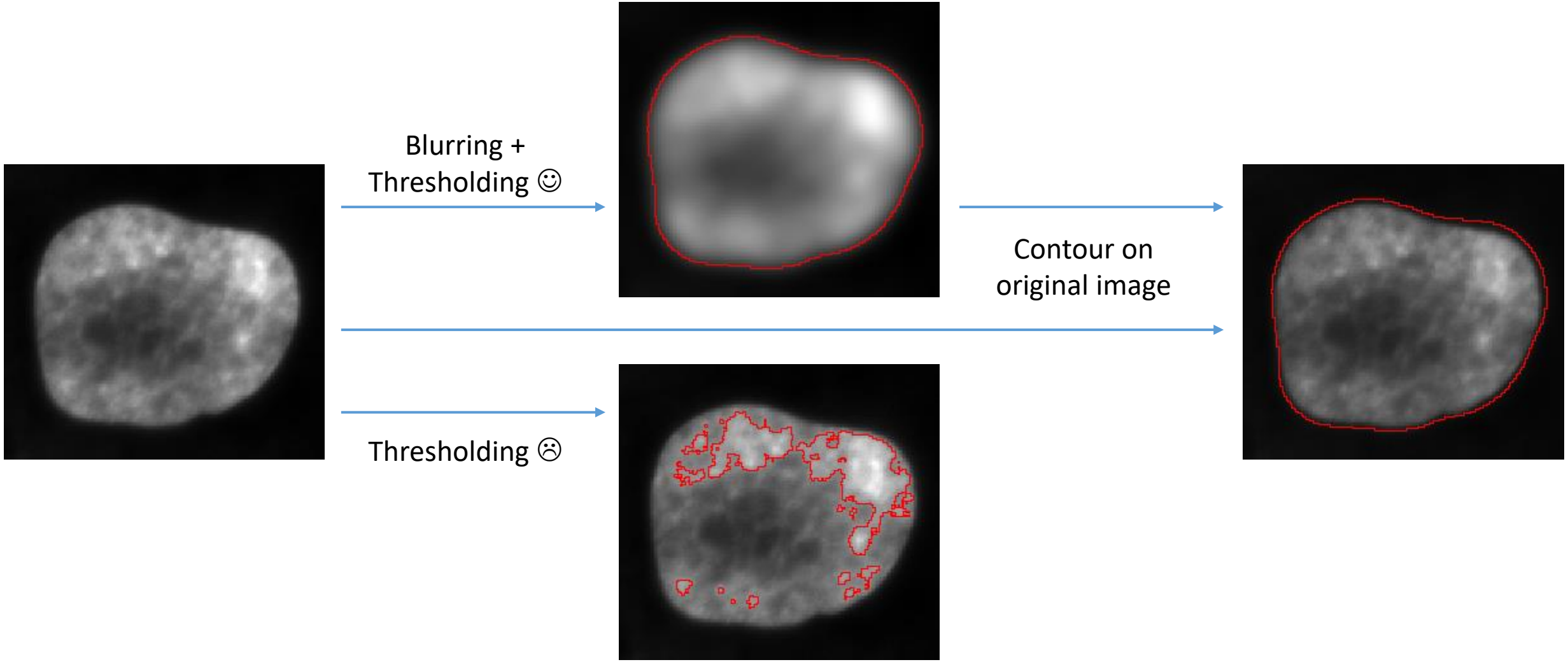


Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

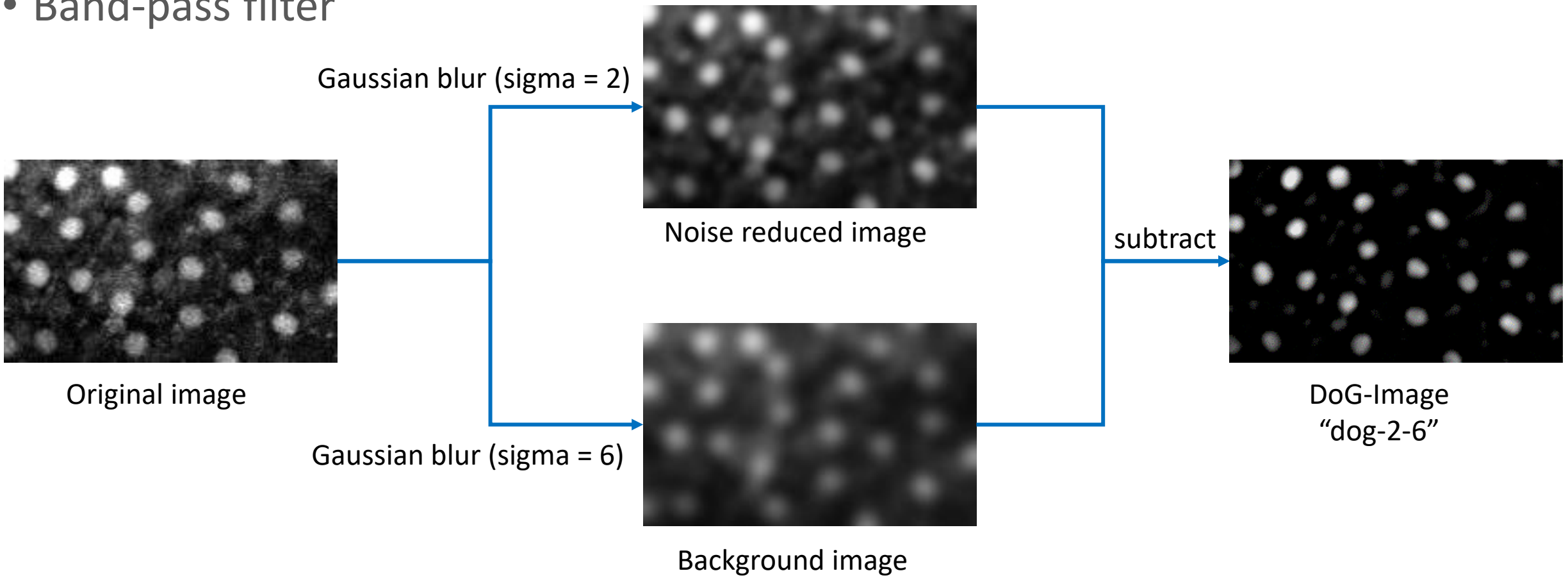
Filtering for improving thresholding results

- In case thresholding algorithms outline the wrong structure, blurring in advance may help.
- However: **Do not** continue processing the blurred image, continue with the original!



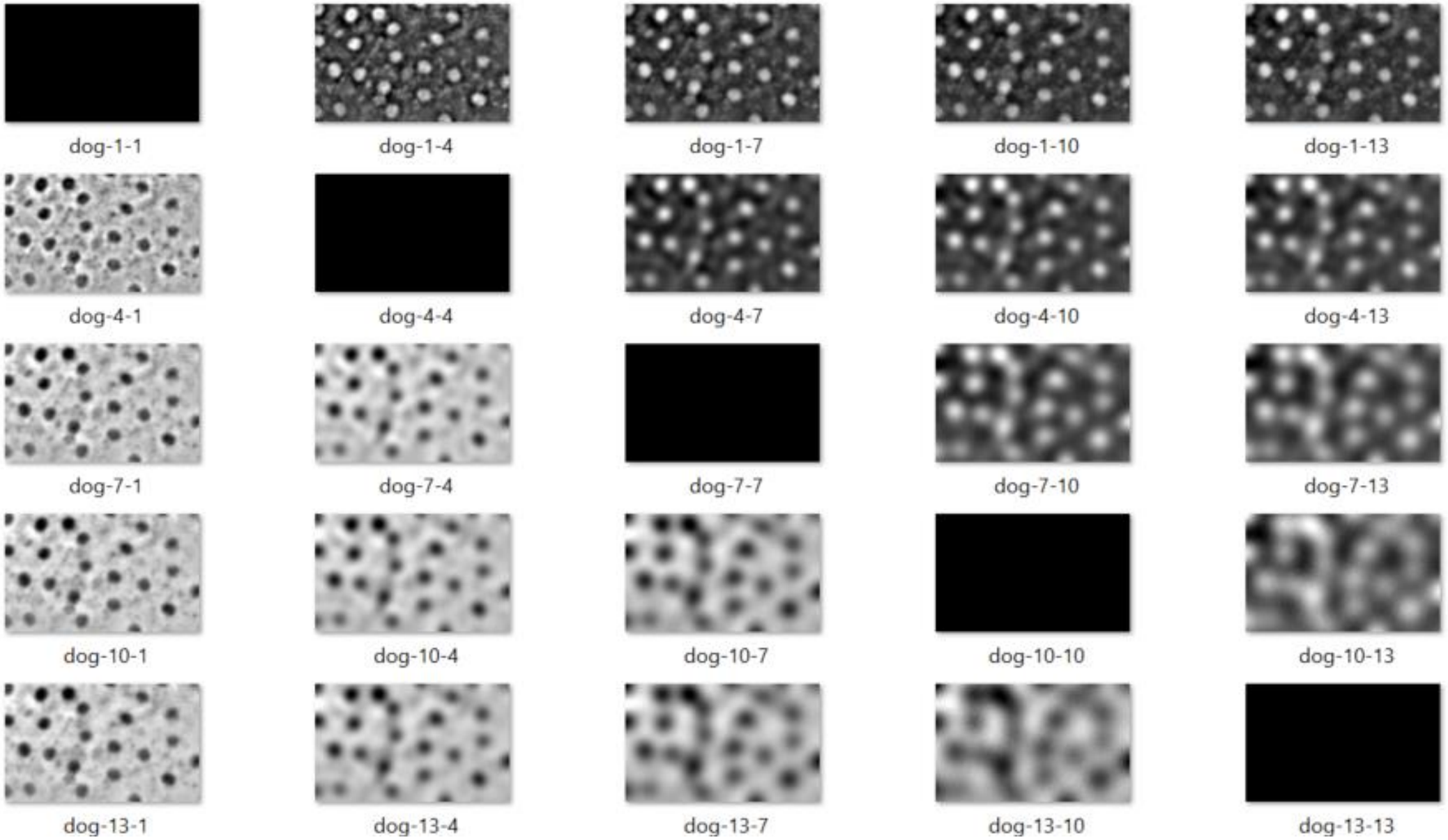
Difference-of-Gaussian (DoG)

- Improve image in order to detect bright objects.
- Band-pass filter



Difference-of-Gaussian (DoG)

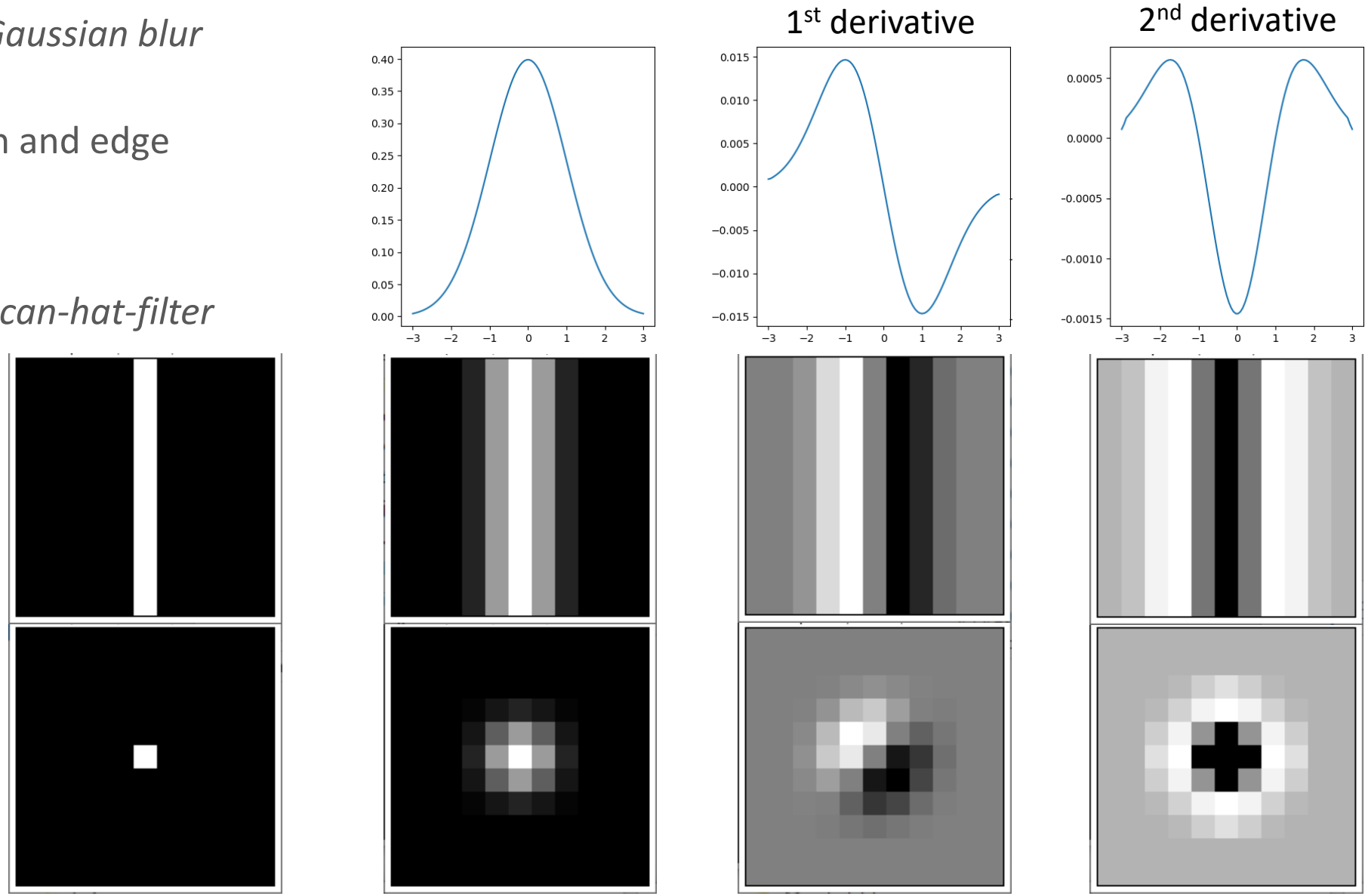
- Example DoG images



Laplace-filter

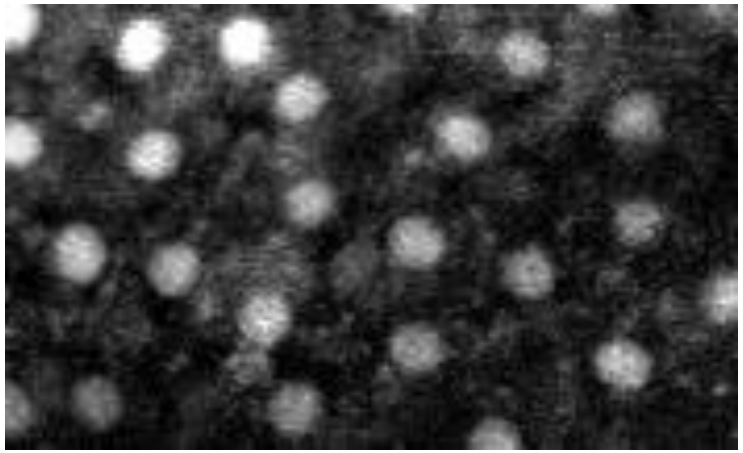
- *Second derivative of a Gaussian blur filter*
- Used for edge-detection and edge enhancement

- Also known as the *Mexican-hat-filter*

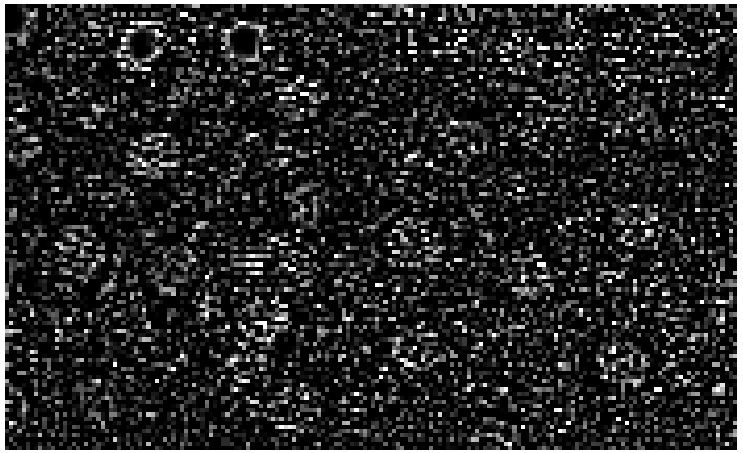


Laplacian-of-Gaussian (LoG)

Laplace filter



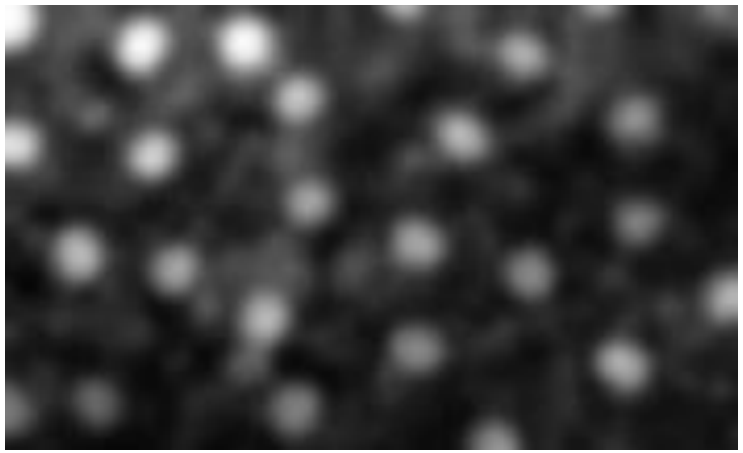
$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$



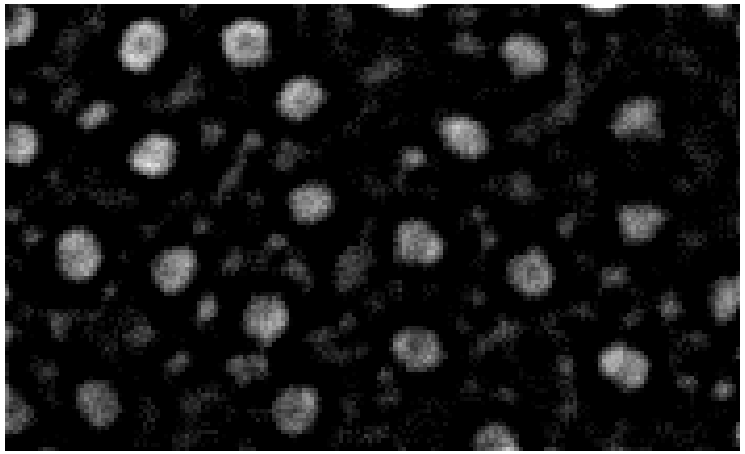
Laplace filtered image

Gaussian filter

Laplacian of Gaussian filter



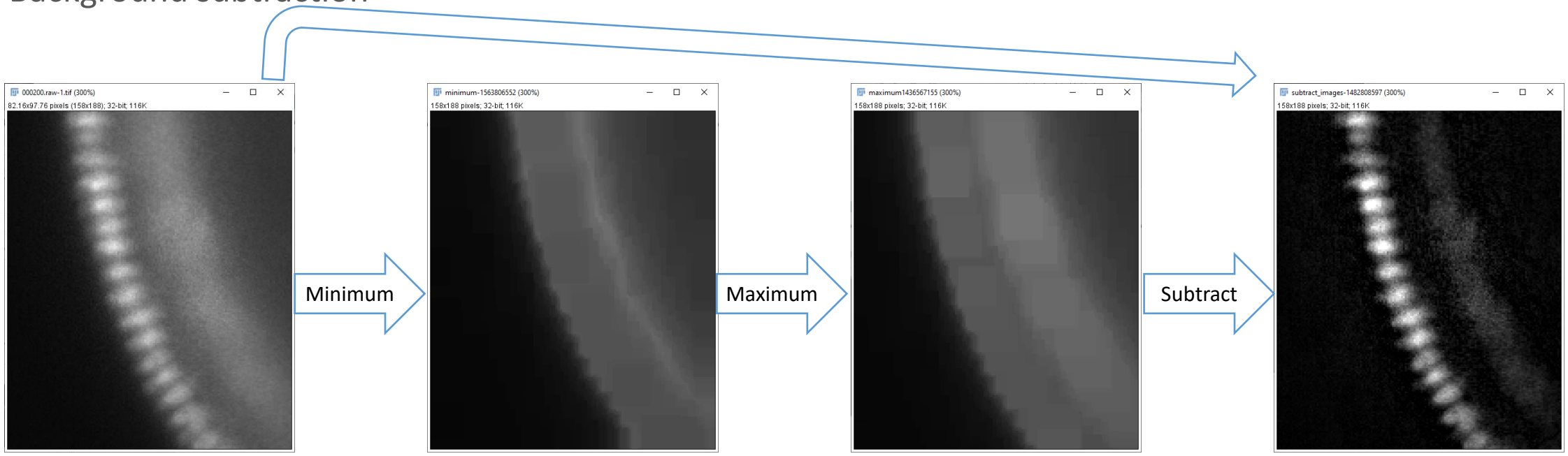
$$* \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} =$$



LoG image

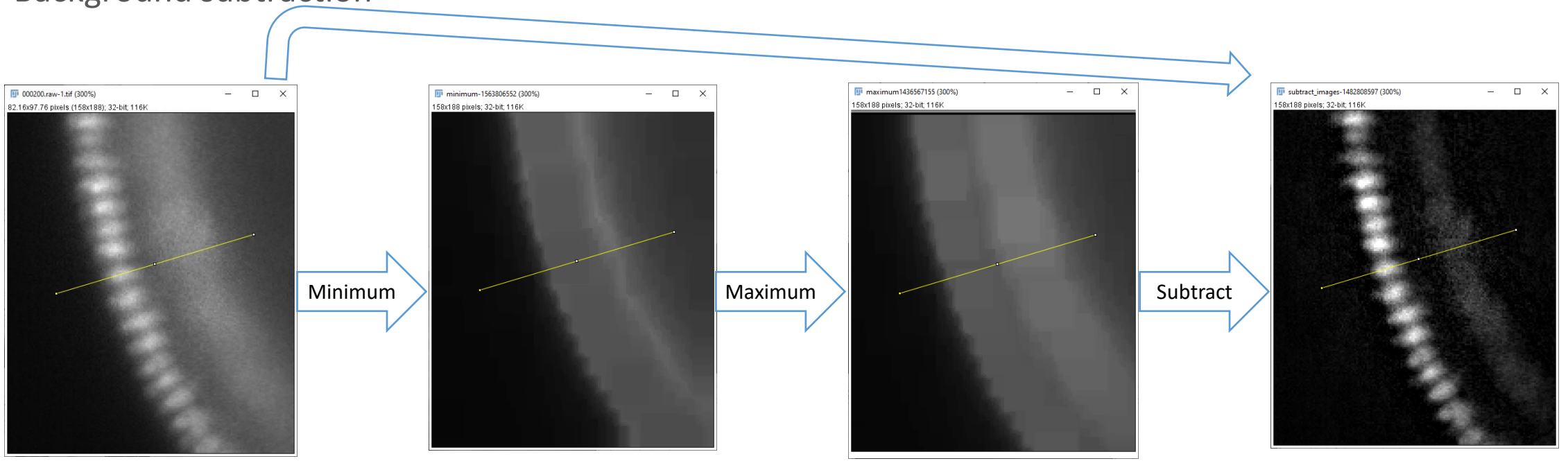
Top-hat filter

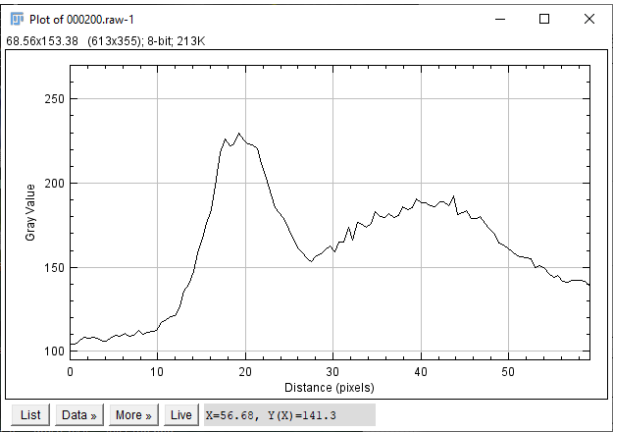
- Background subtraction

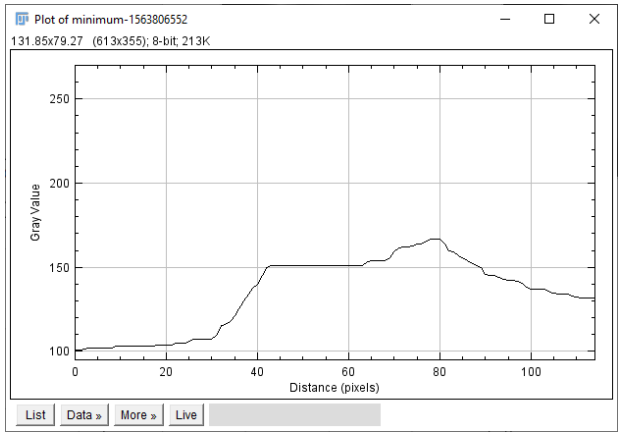


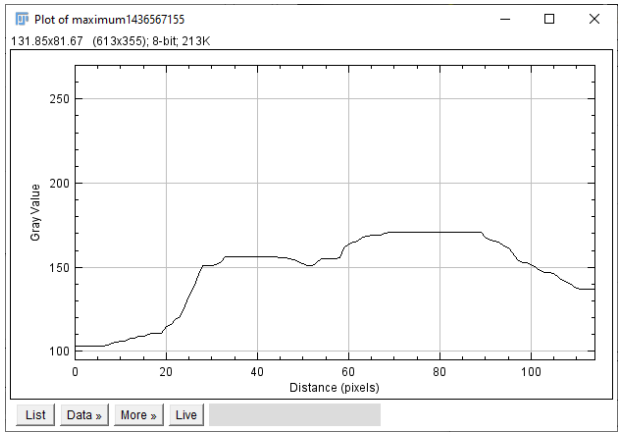
Top-hat filter

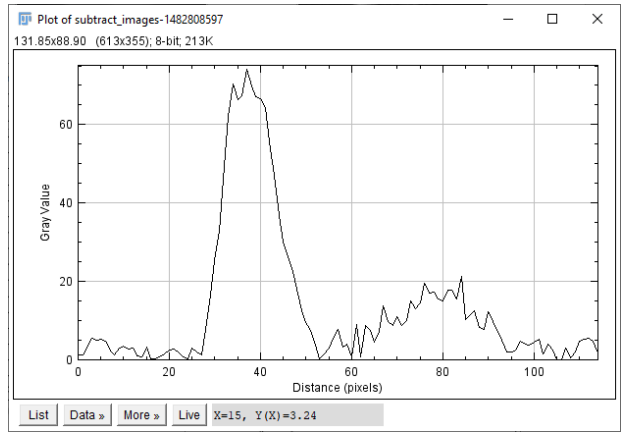
- Background subtraction





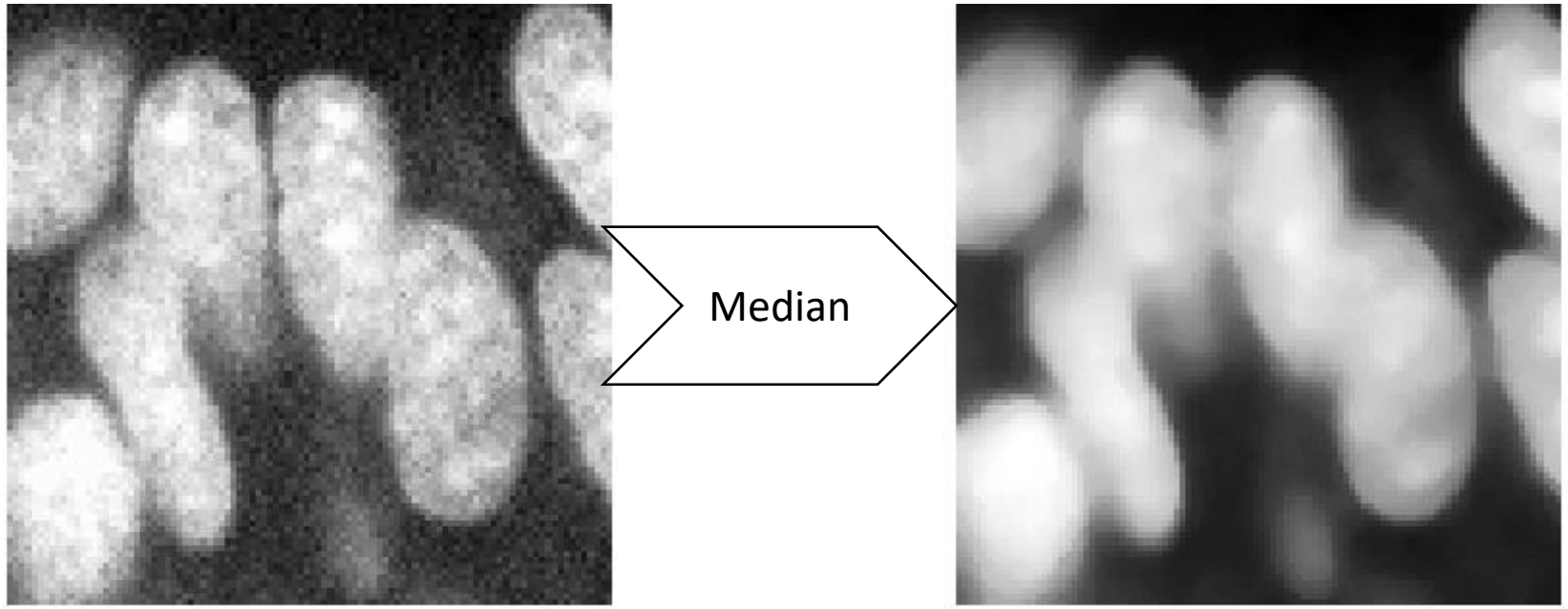






Quiz: Noise removal

- The median filter is a …

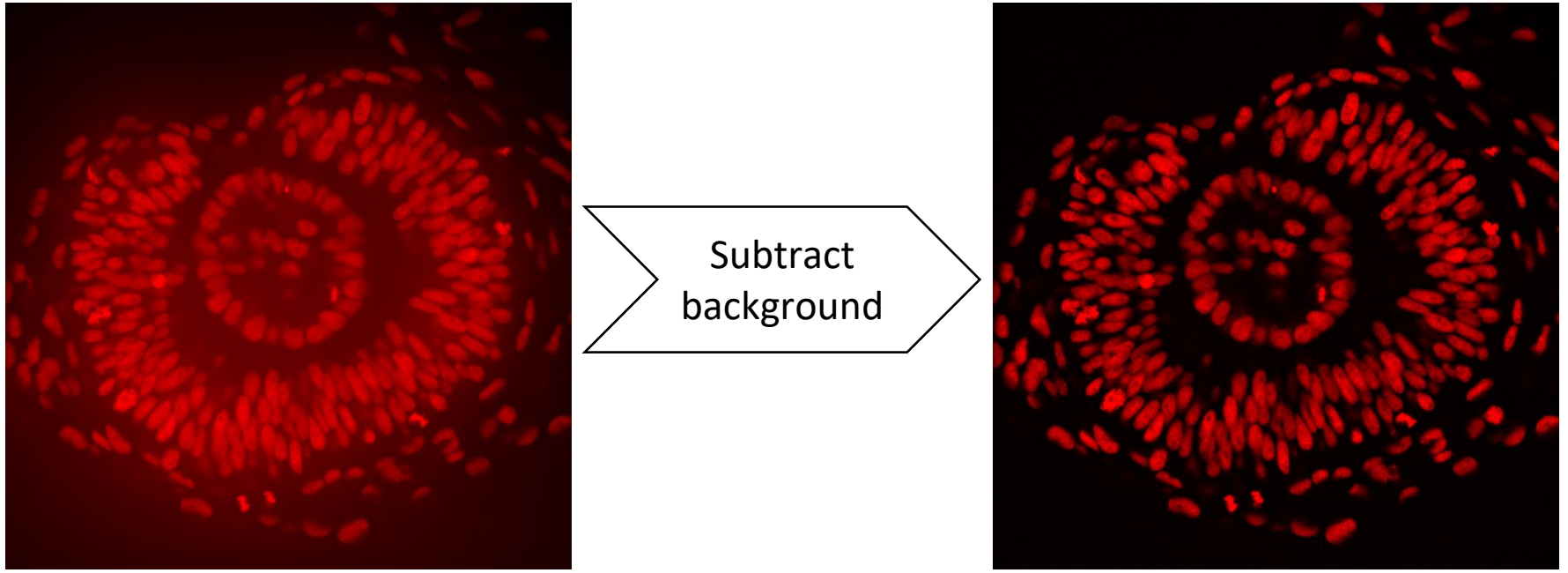


Linear filter

Non-linear filter

Background removal

- Removing background from an image is a … ?



Low-pass filter

High-pass filter

Image source: Mauricio Rocha Martins (Norden/Myers lab, MPI CBG)

PoL
Physics of Life
TU Dresden

CENTER FOR
SYSTEMS BIOLOGY
DRESDEN

Short detour: Image segmentation

Robert Haase

April 2023

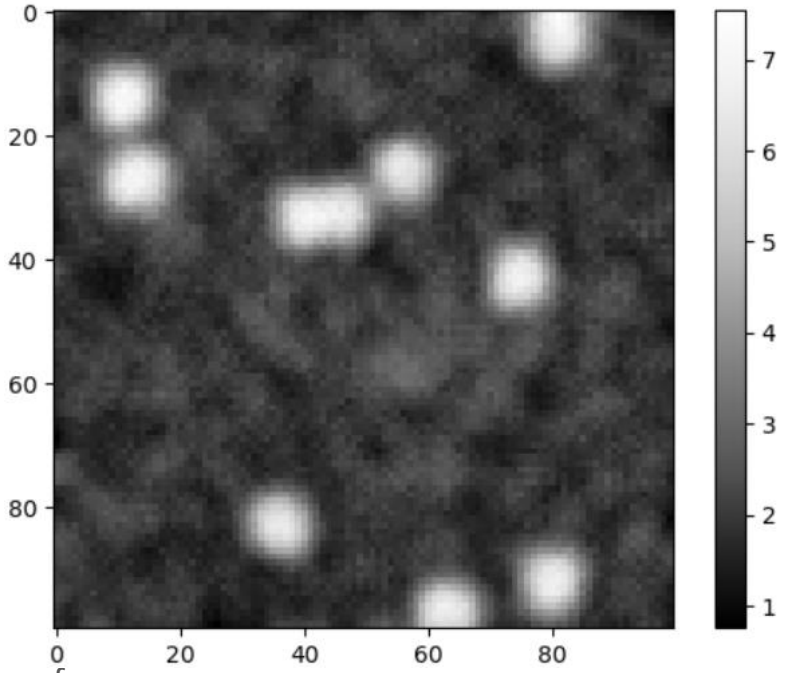
@haesleinhuepf

Short detour: Segmentation

Thresholding

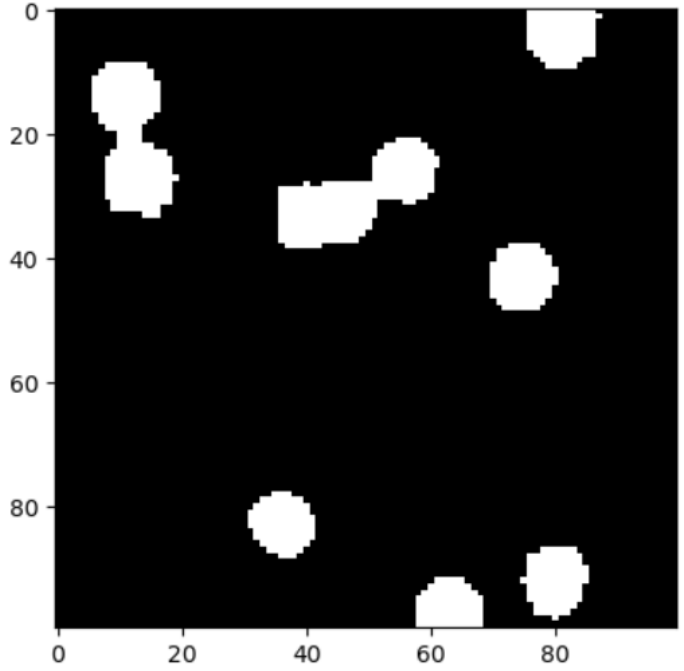
- Very basic and yet efficient segmentation technique
- Histogram based, to determine an intensity threshold
- Not state-of-the-art in many fields (anymore)

Intensity image



?

Binary image

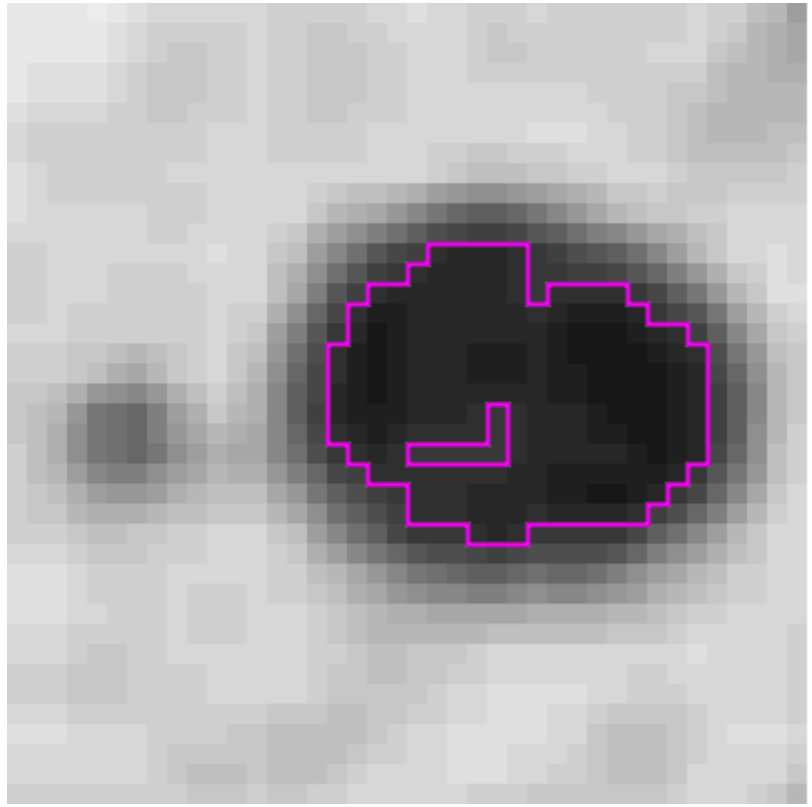


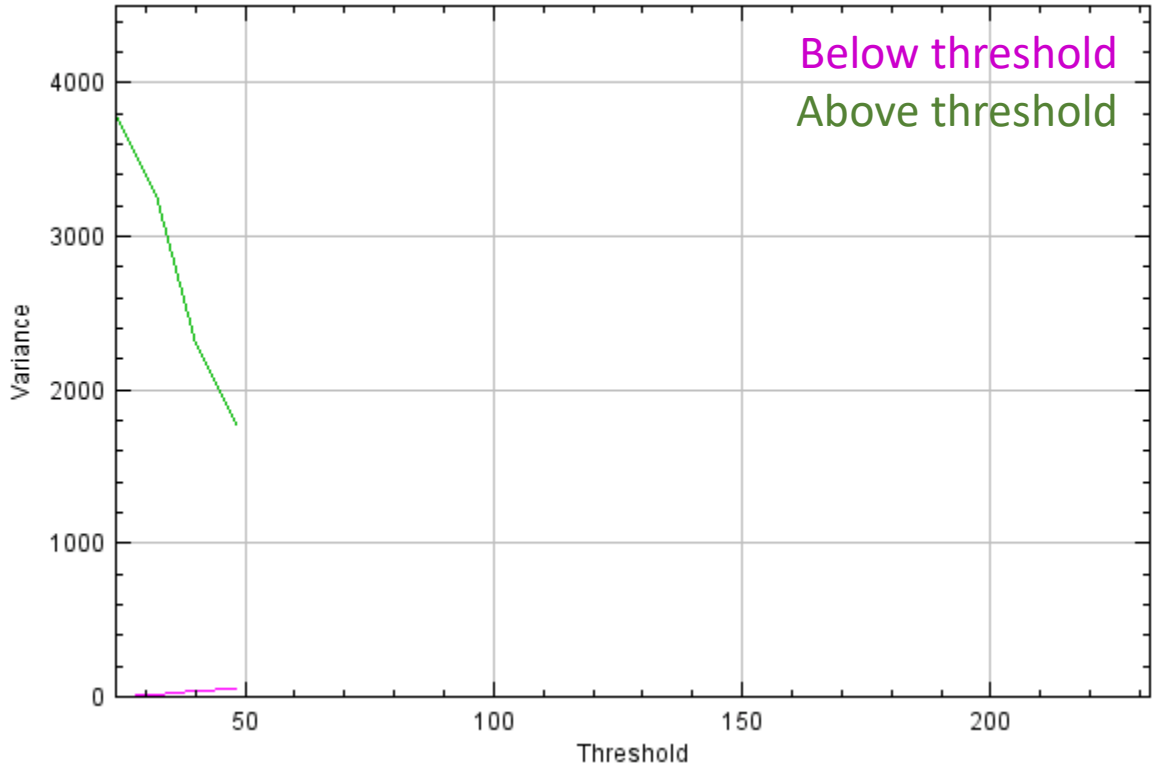
Thresholding: Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.

$$Var(I) = \sum_{i \in I} g_i - \bar{g}_I \qquad \bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$ … Variance in image I
g_i … grey value of a pixel i
$\bar{g}_I$ … mean grey value of the whole image I
n_I … number of pixels in Image I



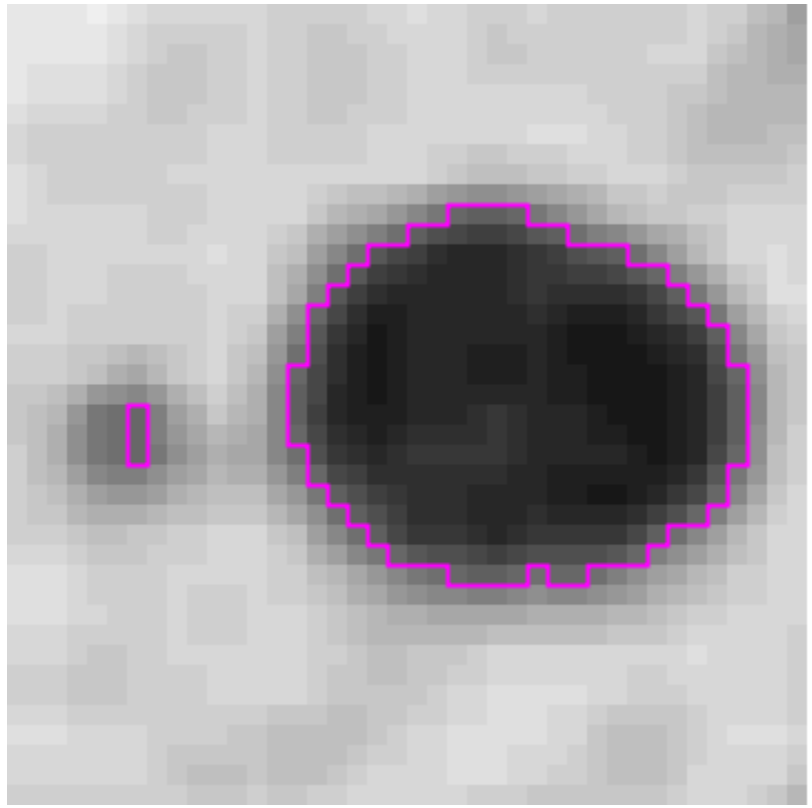


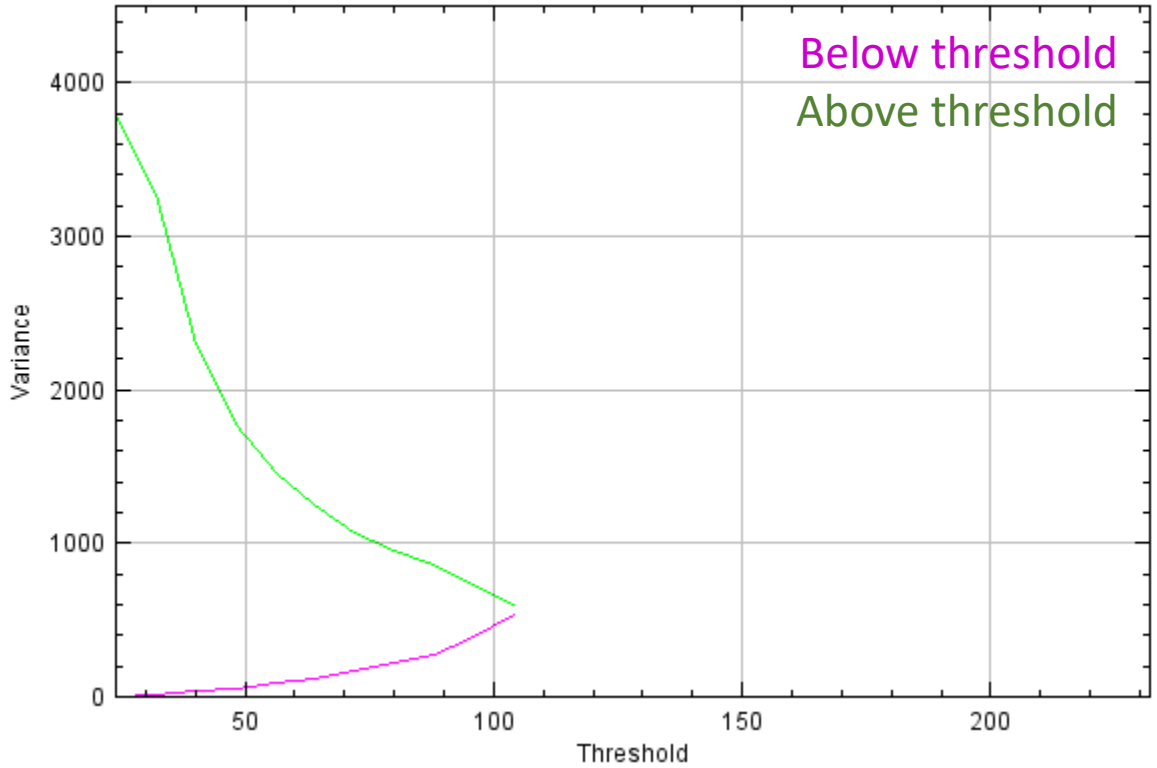
Thresholding: Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.

$$Var(I) = \sum_{i \in I} g_i - \bar{g}_I \qquad \bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$ … Variance in image I
g_i … grey value of a pixel i
$\bar{g}_I$ … mean grey value of the whole image I
n_I … number of pixels in Image I



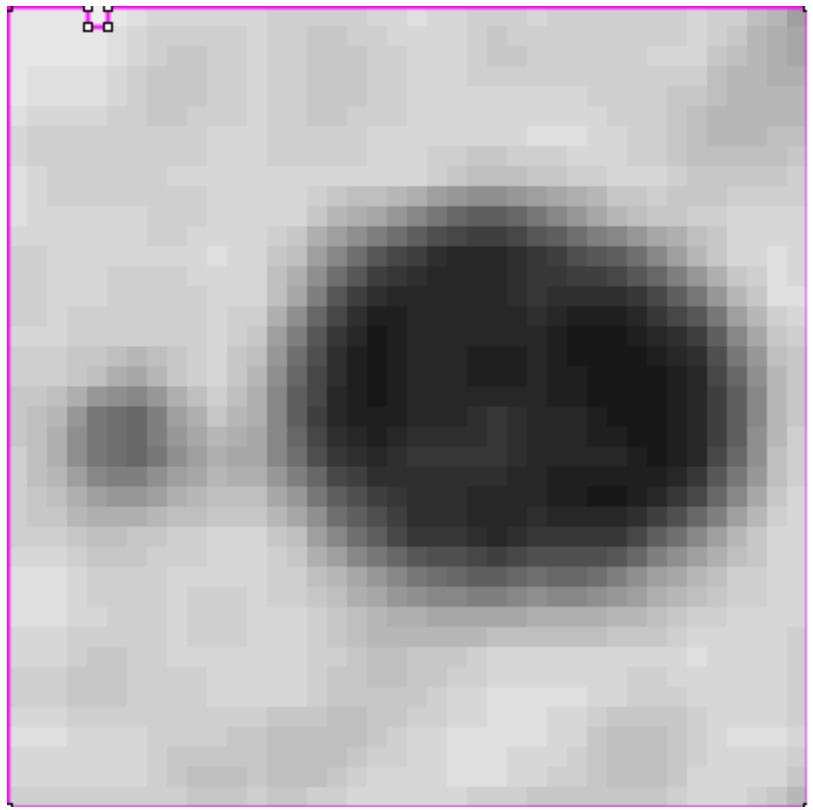


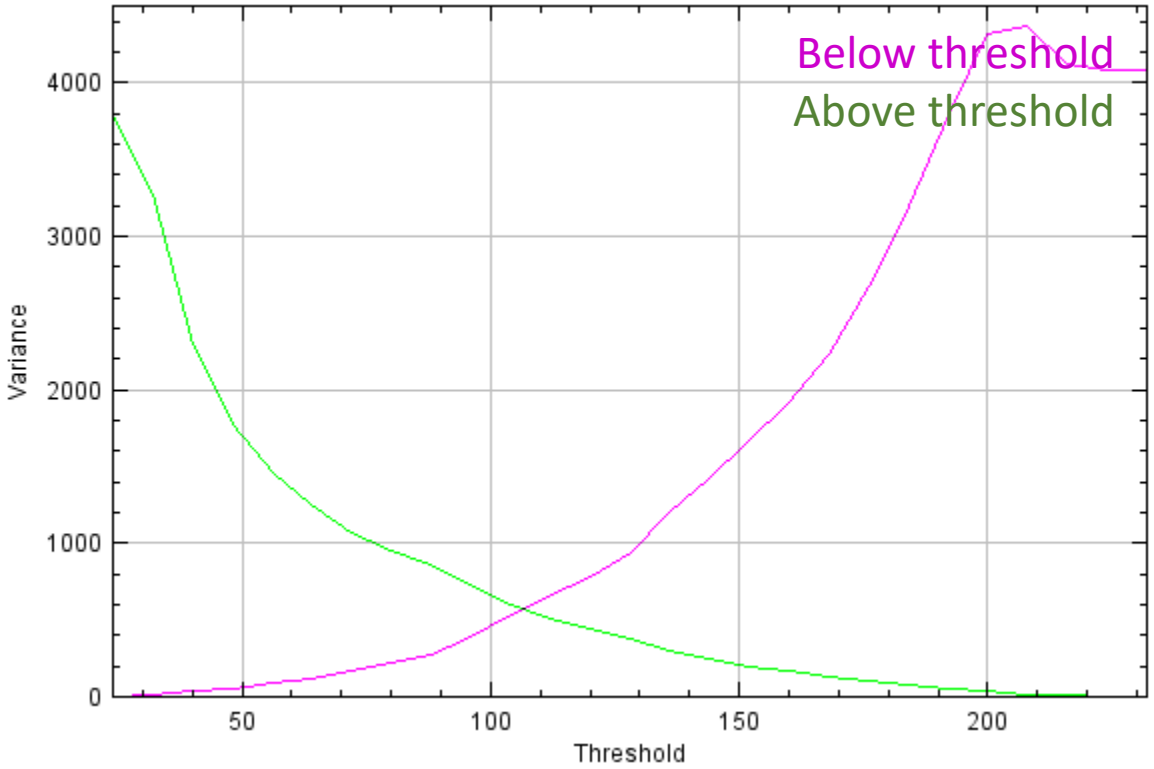
Thresholding: Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.

$$Var(I) = \sum_{i \in I} g_i - \bar{g}_I \qquad \bar{g}_I = \sum_{i \in I} \frac{g_i}{n_I}$$

$Var(I)$ … Variance in image I
g_i … grey value of a pixel i
$\bar{g}_I$ … mean grey value of the whole image I
n_I … number of pixels in Image I

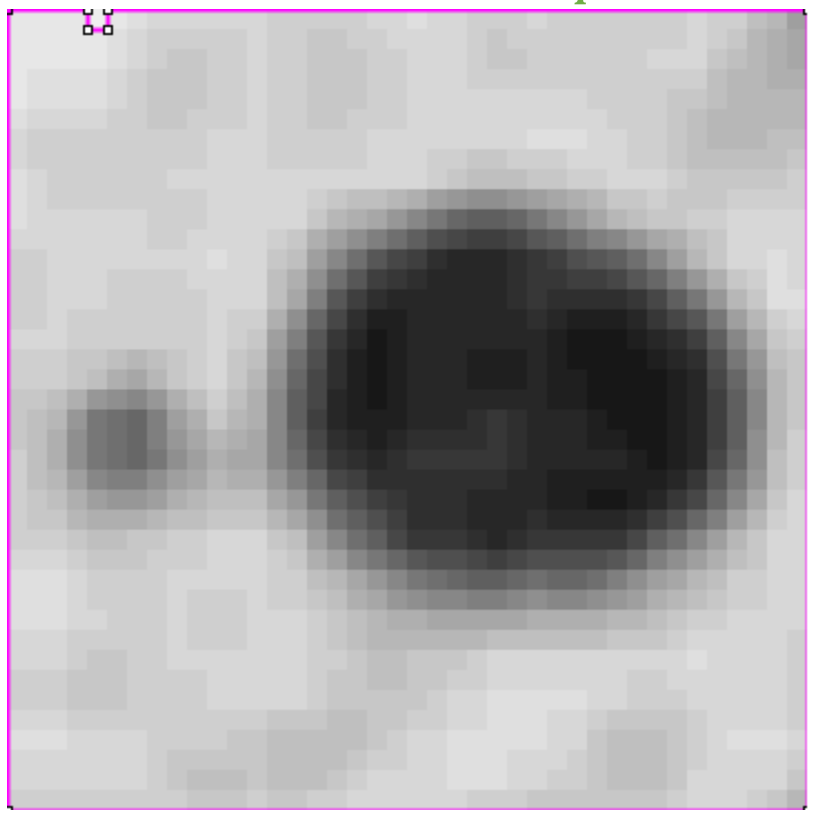


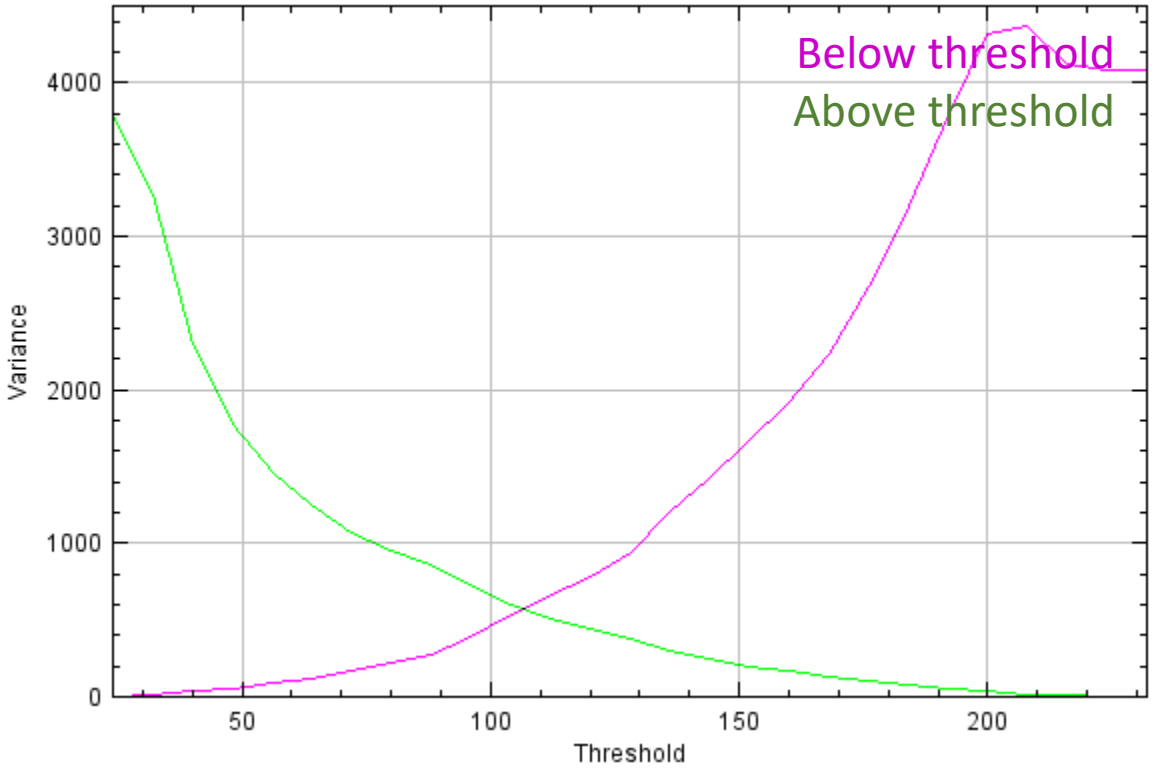


Thresholding: Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.
- Weighted (!) sum variance

$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \qquad I = A \cup B$$

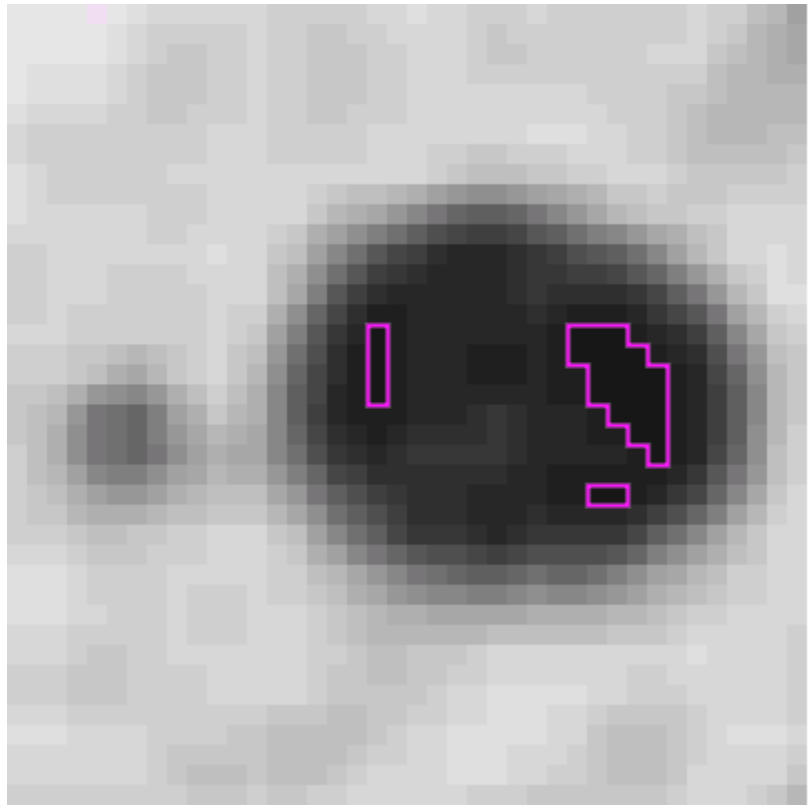


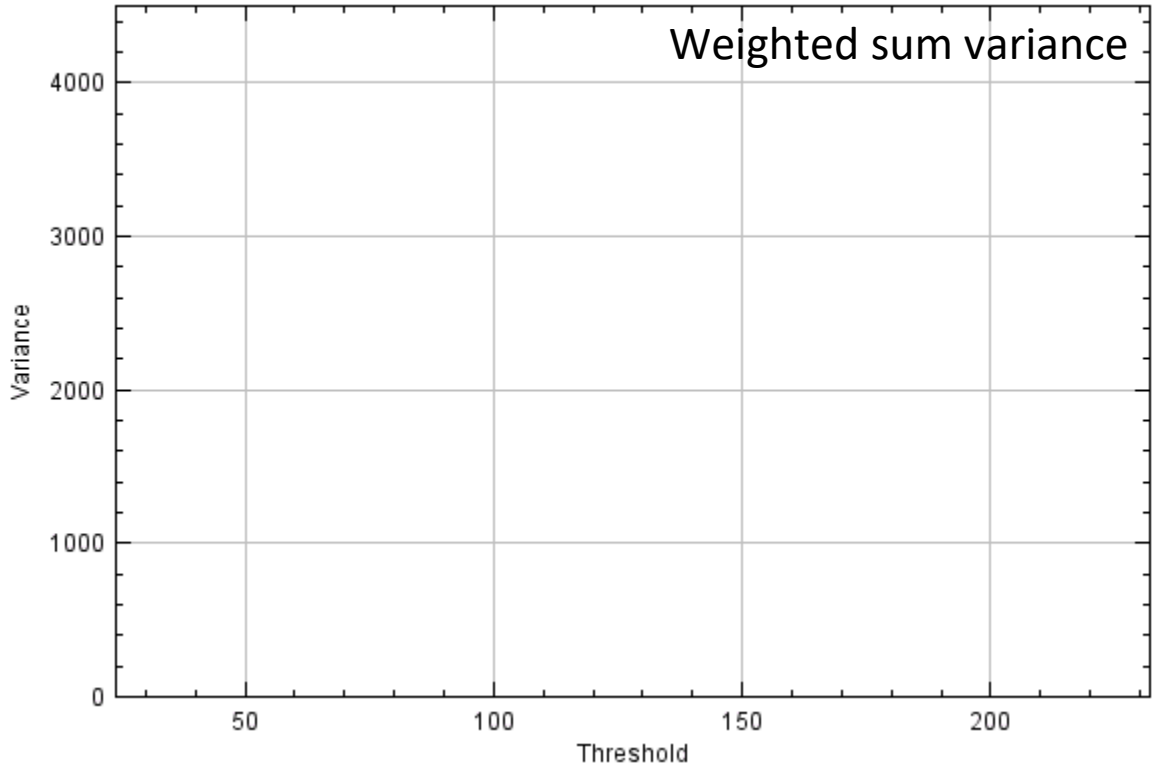


Thresholding: Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.
- Weighted (!) sum variance

$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \qquad I = A \cup B$$

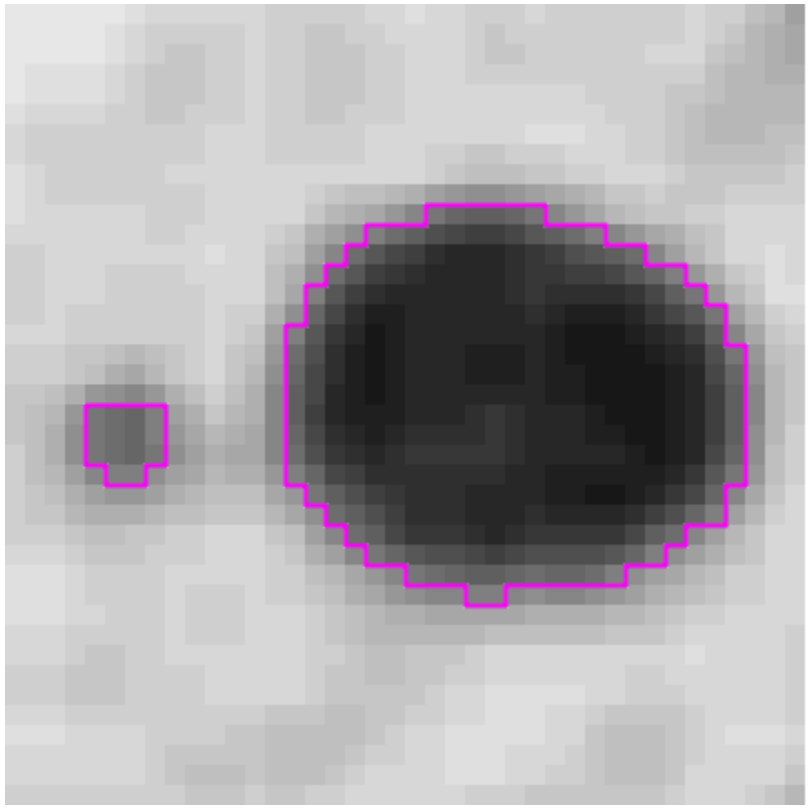


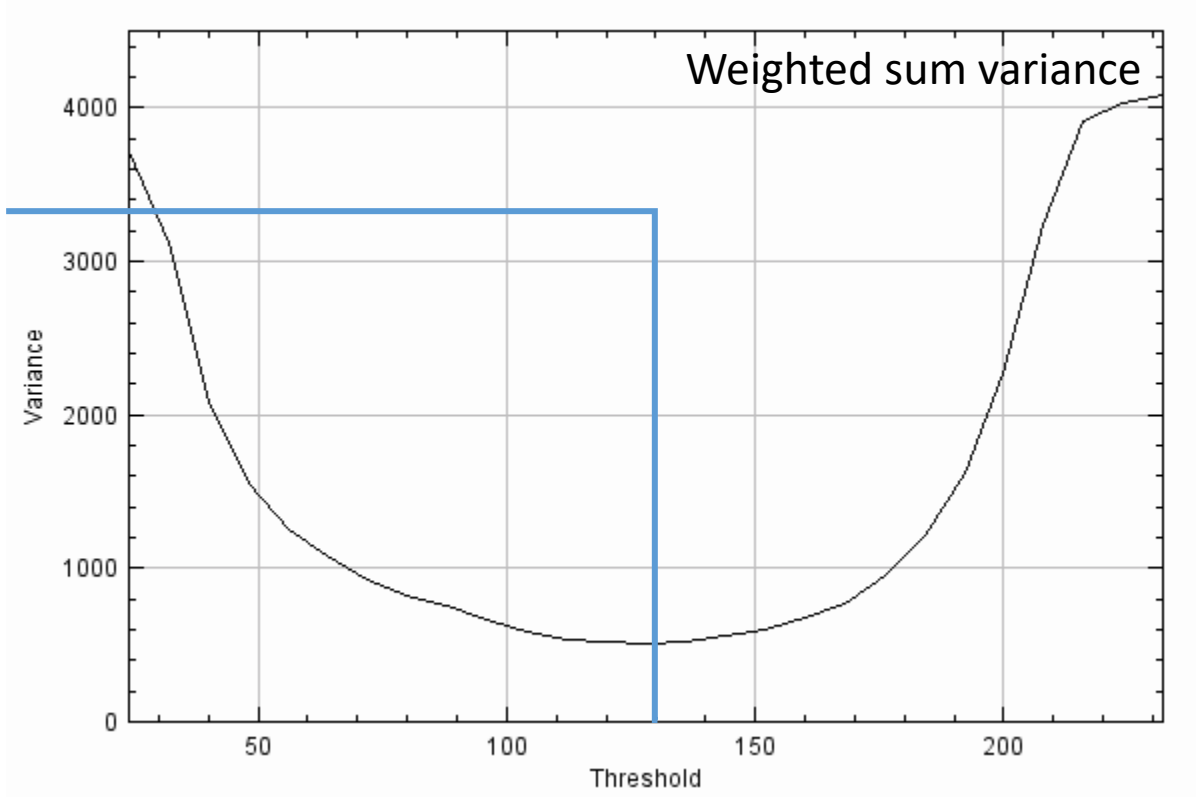


Thresholding: Otsu's method

- Searching for a threshold where the variance in both classes (above/below threshold) becomes minimal.
- Weighted (!) sum variance

$$Var'(I) = \frac{n_A}{n_I} Var(A) + \frac{n_B}{n_I} Var(B) \qquad I = A \cup B$$





See also: http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html

Thresholding: Citing

- Cite the thresholding method of your choice properly

"We segmented the cell nuclei in the images using Otsu's thresholding method (Otsu et Al. 1979) implemented in scikit-image (van der Walt et al. 2014)."

IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, VOL. SMC-9, NO. 1, JANUARY 1979

A Threshold Selection Method from Gray-Level Histograms

NOBUYUKI OTSU

Abstract—**A nonparametric and unsupervised method of automatic threshold selection for picture segmentation is presented. An optimal threshold is selected by the discriminant criterion, namely, so as to maximize the separability of the resultant classes in gray levels. The procedure is very simple, utilizing only the zeroth- and the first-order cumulative moments of the gray-level histogram. It is straightforward to extend the method to multithreshold problems. Several experimental results are also presented to support the validity of the method.**

PoL Physics of Life TU Dresden

CENTER FOR SYSTEMS BIOLOGY DRESDEN

Image Processing: Morphological Operations

Robert Haase

With material from

Marcelo Leomil Zoccoler, Physic of Life, TU Dresden

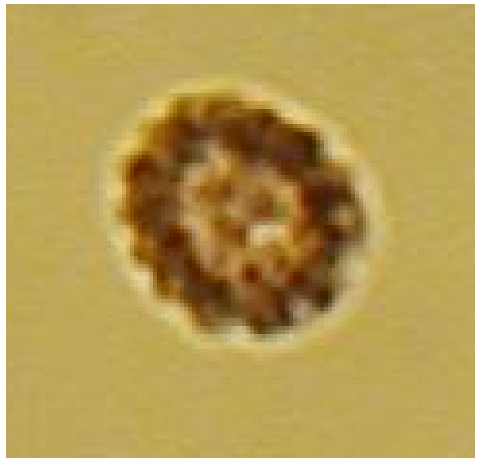
April 2023

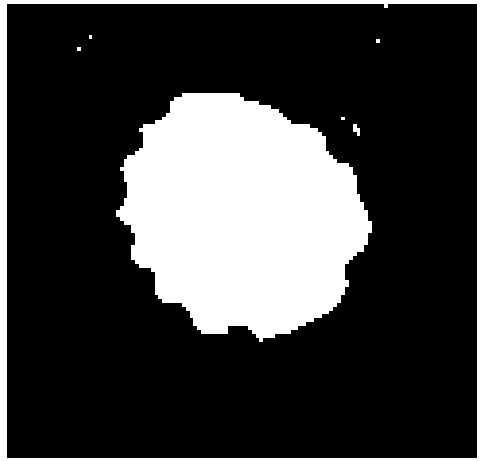
@haesleinhuepf

Refining masks

- Binary mask images may not be perfect immediately after thresholding.
- There are ways of refining them

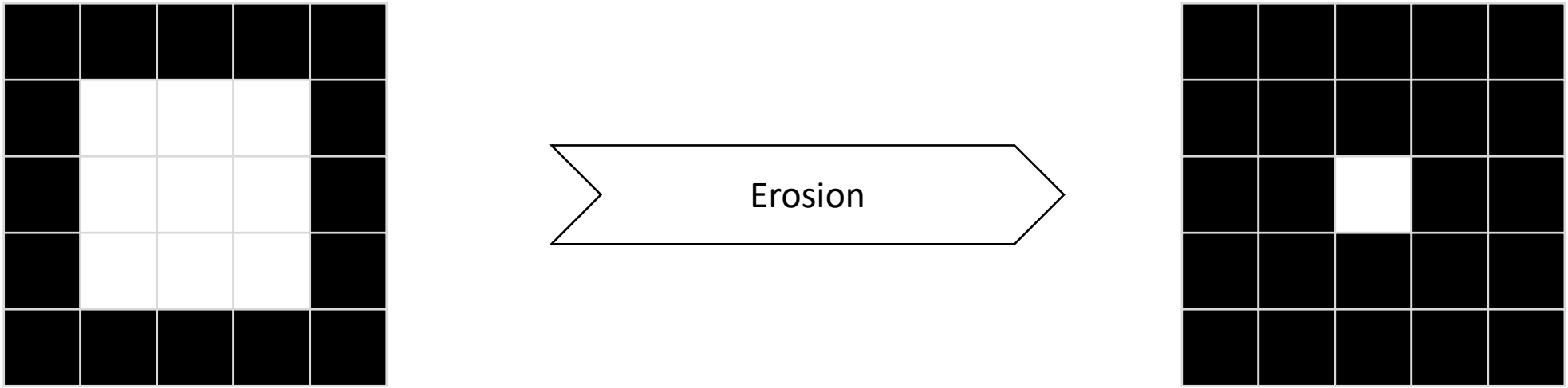
Thresholding → Closing → Opening





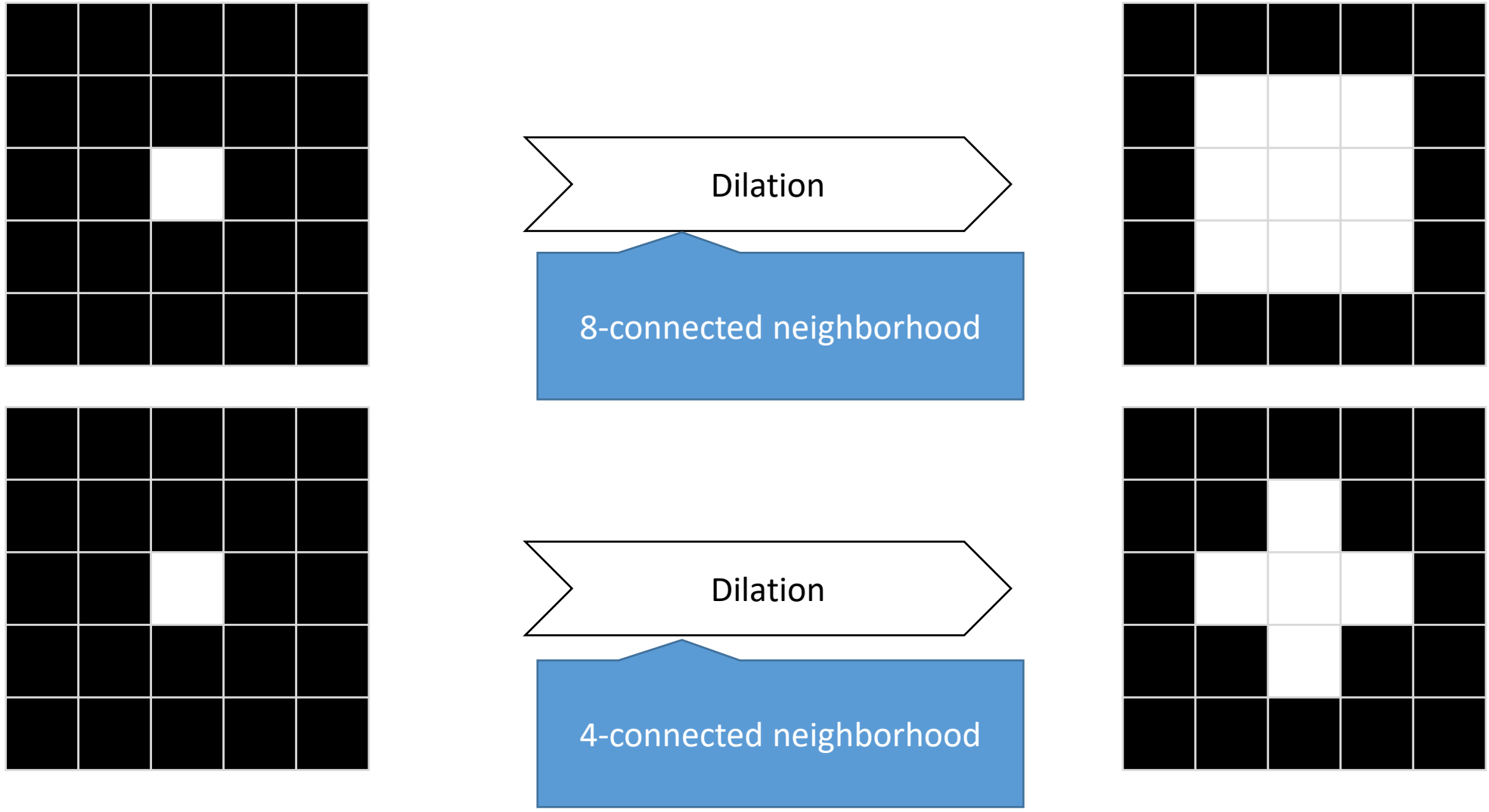

Erosion

- Erosion: Every pixel with at least one black neighbor becomes black.



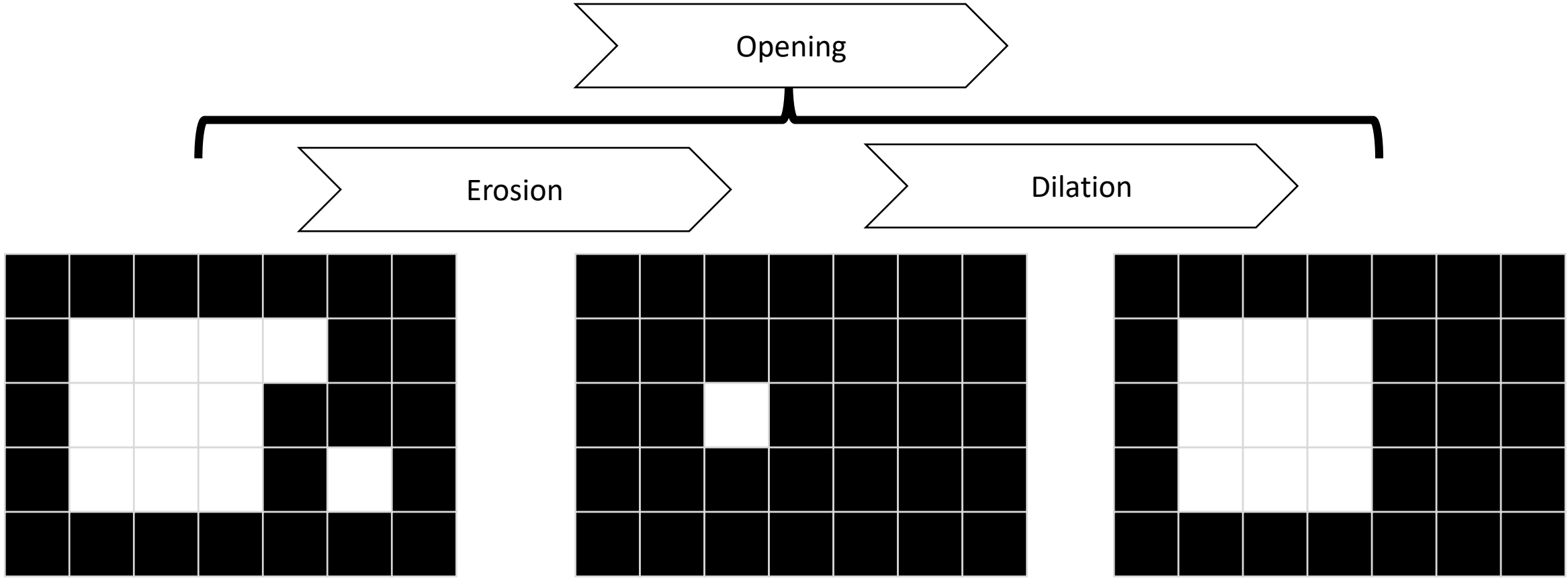
Dilation

- Dilation: Every pixel with at least one white neighbor becomes white.



Opening

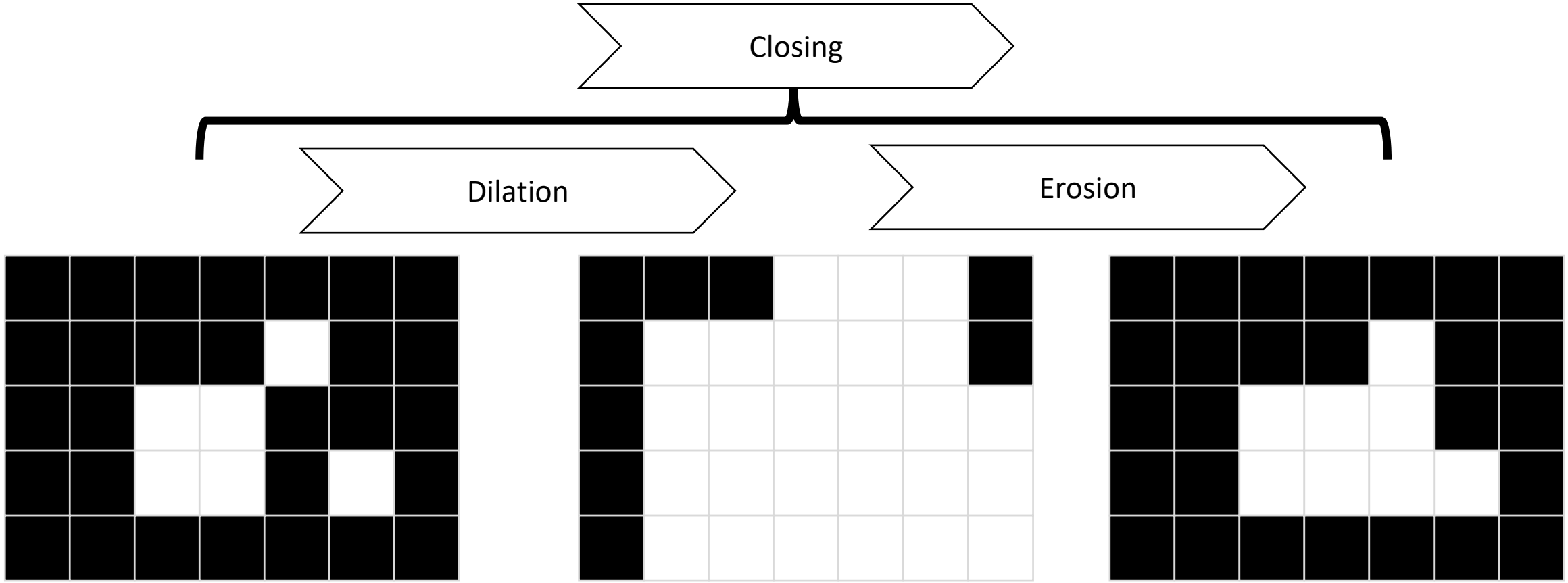
- Erosion and dilation combined allow correcting outlines.



- It can separate white (high intensity) structures that are weakly connected
- It may erase small white structures

Closing

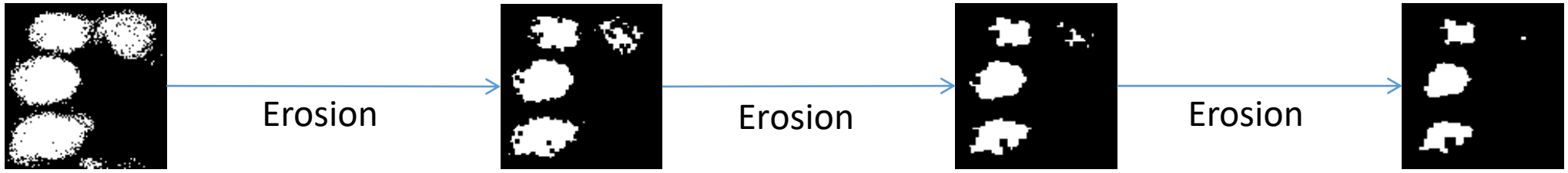
- Erosion and dilation combined allow correcting outlines.



- It can connect white (high intensity) structures that are nearby
- It may close small holes inside structures

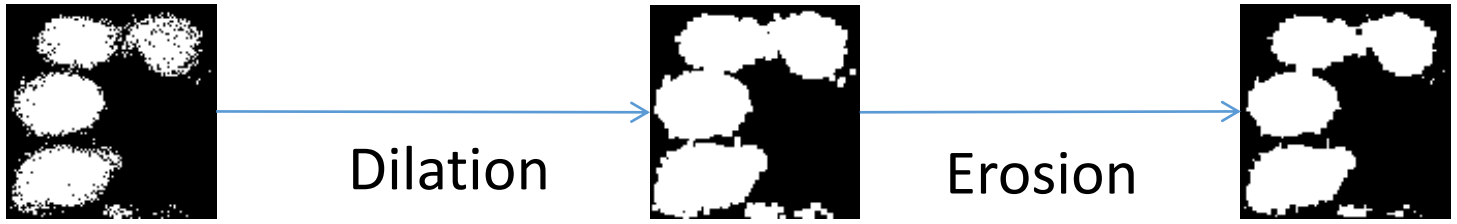
Chaining erosion and dilation

- Erosion: Set all pixels to black which have at least one black neighbor.



- Dilation: Set all pixels to white which have at least one white neighbor.

- Closing: Dilation + Erosion



- Opening: Erosion + Dilation

PoL
Physics of Life
TU Dresden

CENTER FOR
SYSTEMS BIOLOGY
DRESDEN

Image Processing in Python

Robert Haase

With material from

Marcelo Leomil Zoccoler, Physics of Life, TU Dresden

April 2023

Working with images in python

PoL Physics of Life TU Dresden

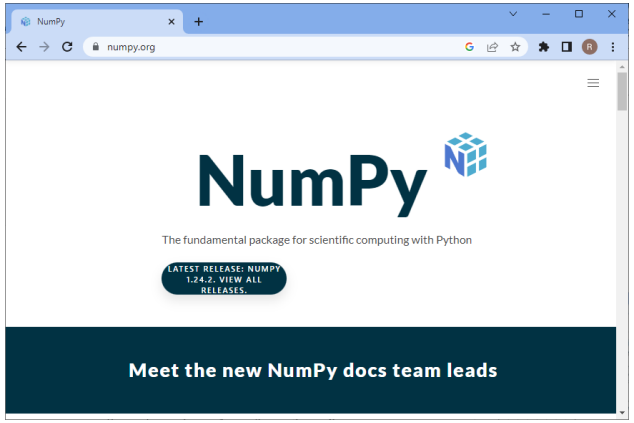
- Open images

```
from skimage.io import imread

image = imread("blobs.tif")
```

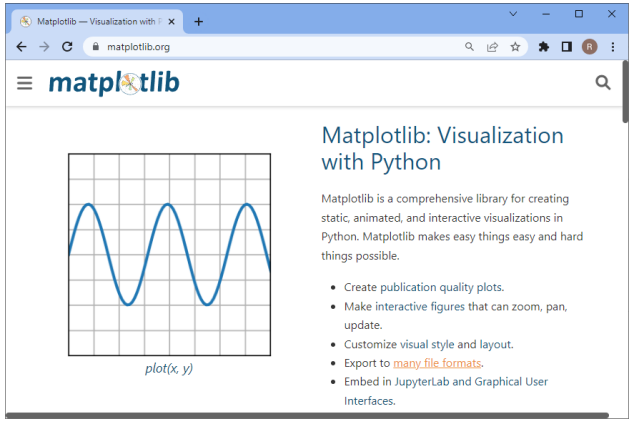
```
image
```

```
array([[ 40,  32,  24, ..., 216, 200, 200],
       [ 56,  40,  24, ..., 232, 216, 216],
       [ 64,  48,  24, ..., 240, 232, 232],
       ...,
       [ 72,  80,  80, ...,  48,  48,  48],
       [ 80,  80,  80, ...,  48,  48,  48],
       [ 96,  88,  80, ...,  48,  48,  48]], dtype=uint8)
```

Images are *just* multi-dimensional arrays or "arrays of arrays".



https://numpy.org/



https://matplotlib.org/

Working with images in python

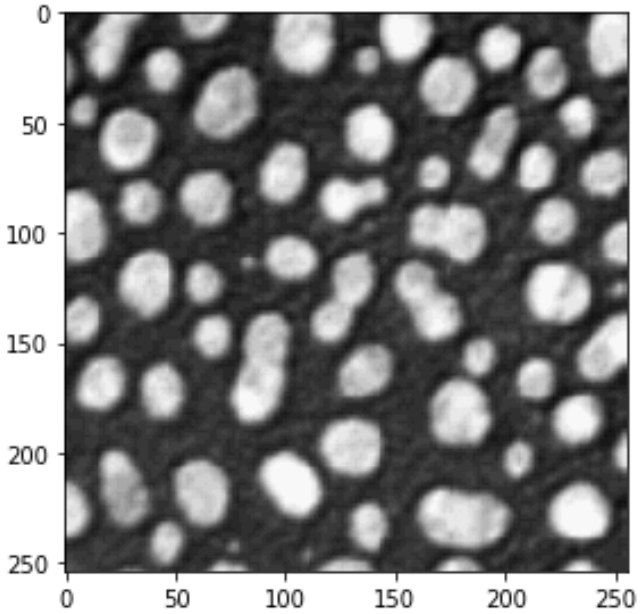
- Open images

```
from skimage.io import imread

image = imread("blobs.tif")
```

- Visualize images

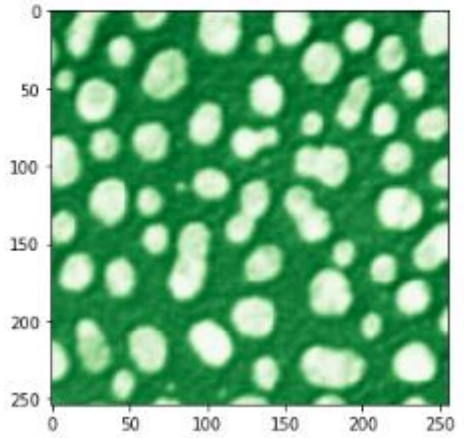
```
from skimage.io import imshow

imshow(image)
```

```
<matplotlib.image.AxesImage at 0x245e7
```



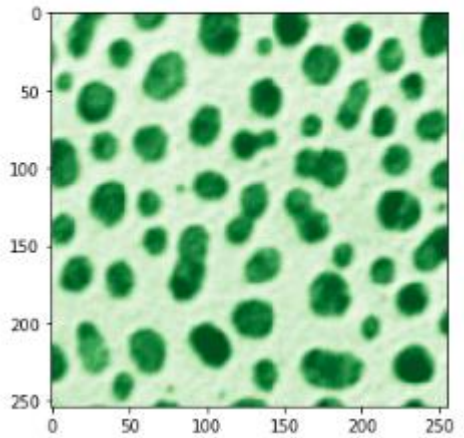

```
imshow(image, cmap="Greens_r")
```

```
<matplotlib.image.AxesImage at 0
```



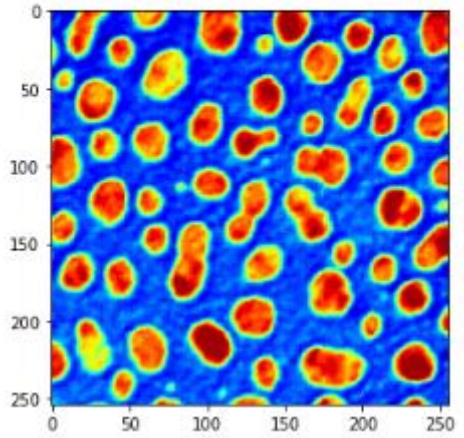

```
imshow(image, cmap="Greens")
```

```
<matplotlib.image.AxesImage at
```




```
imshow(image, cmap="jet")
```

```
<matplotlib.image.AxesImage at
```



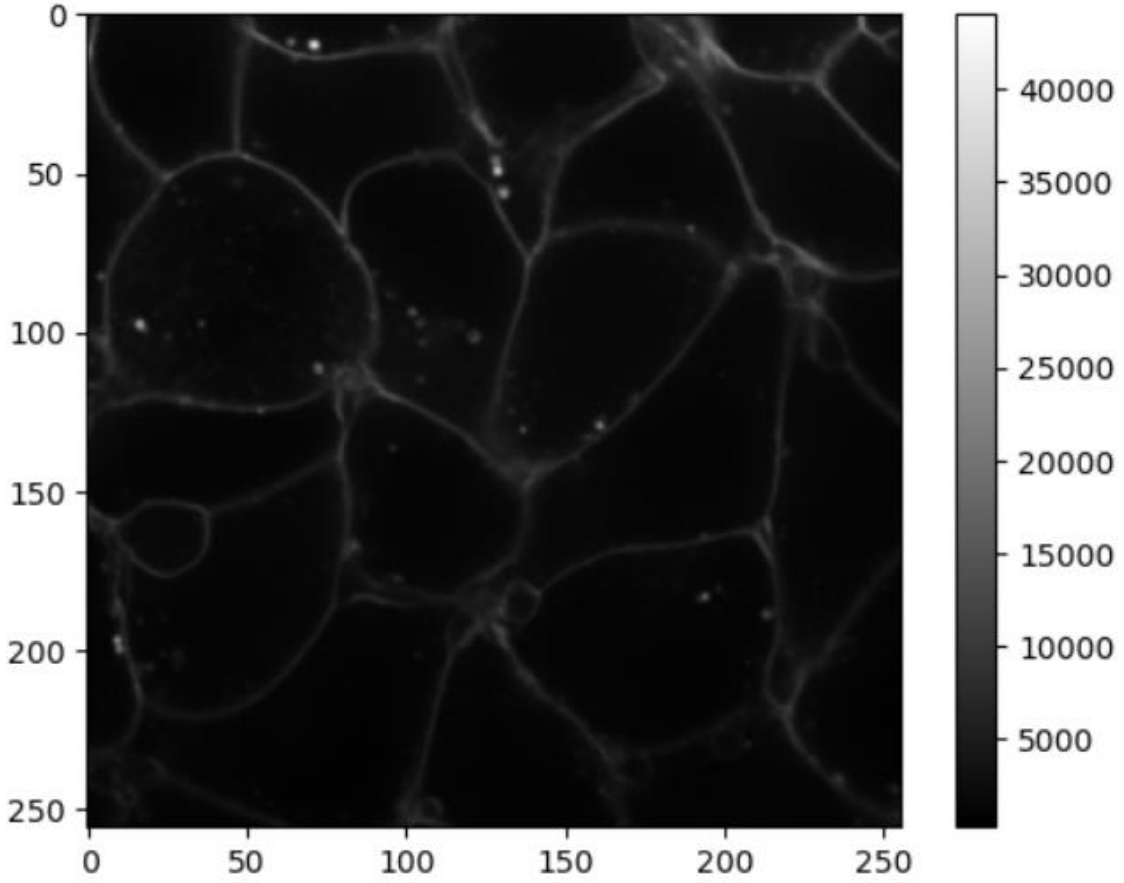
This does not modify the image data. The images are just shown with different colors representing the same values.

Brightness, contrast, display-range

- After loading data, make sure you can see the structure you're interested in

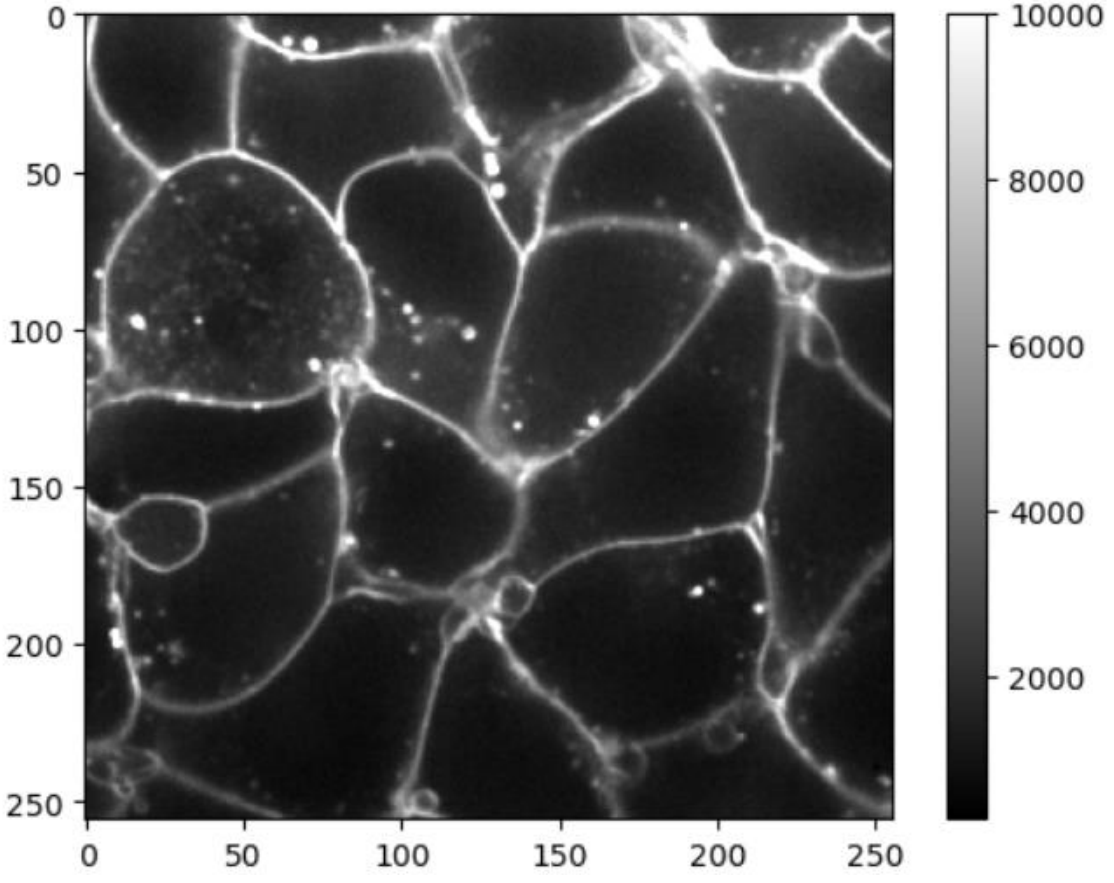
```
plt.imshow(image, cmap='gray')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x14f22cf71f0>
```




```
plt.imshow(image, cmap='gray', vmax=10000)
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x14f22d70310>
```

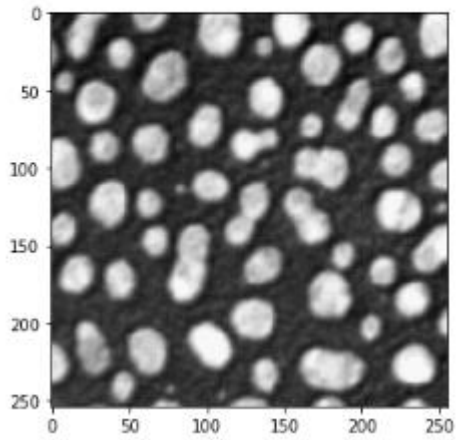


Cropping and resampling images

- Indexing and cropping *numpy-arrays* works like with python arrays.

```
imshow(image)
```

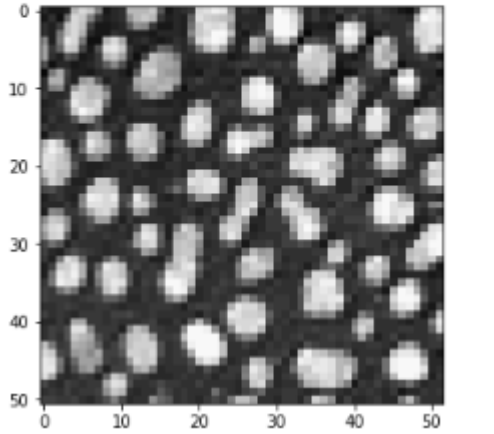
```
<matplotlib.image.AxesImage at 0
```



Original image

```
sampled_image = image[::5, ::5]

imshow(sampled_image)
```

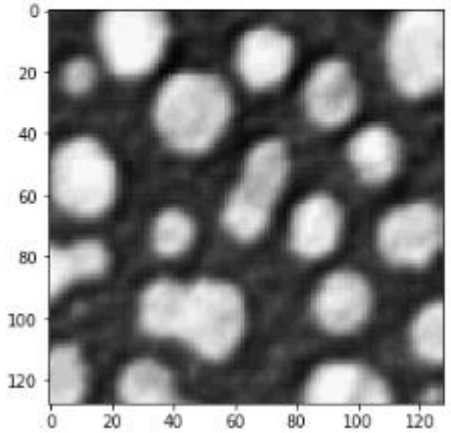
```
<matplotlib.image.AxesImage at 0
```



Sub-sampled image

```
cropped_image2 = image[0:128, 128:]

imshow(cropped_image2)
```

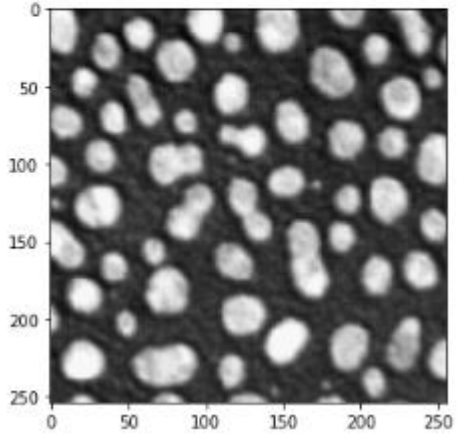
```
<matplotlib.image.AxesImage at 0x29e
```



Cropped image

```
flipped_image = image[:, ::-1]

imshow(flipped_image)
```

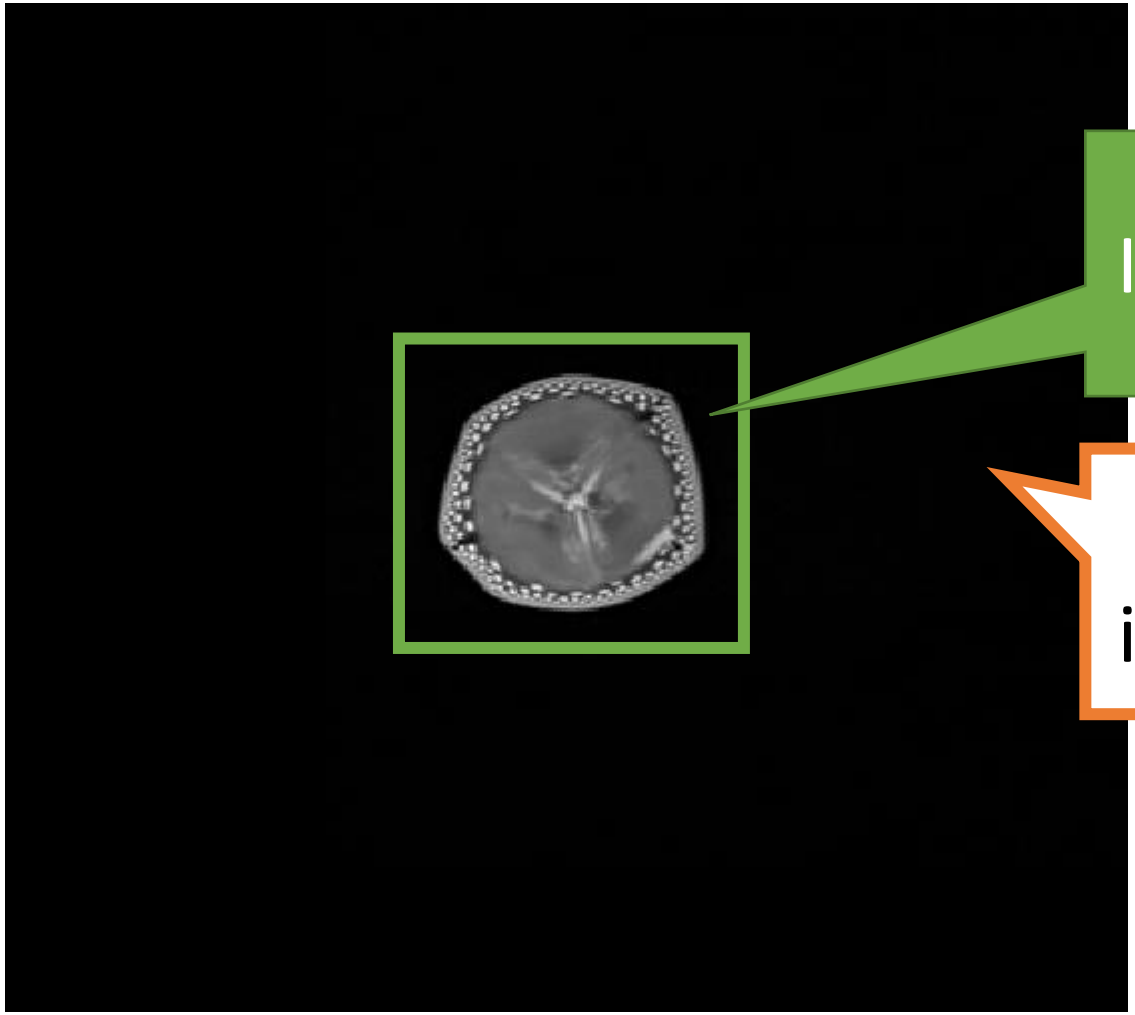
```
<matplotlib.image.AxesImage at 0x
```



Flipped image

Cropping and resampling images

- Crop out the region you're interested in



In this case
you can spare
8/9 compute time for
following processing steps

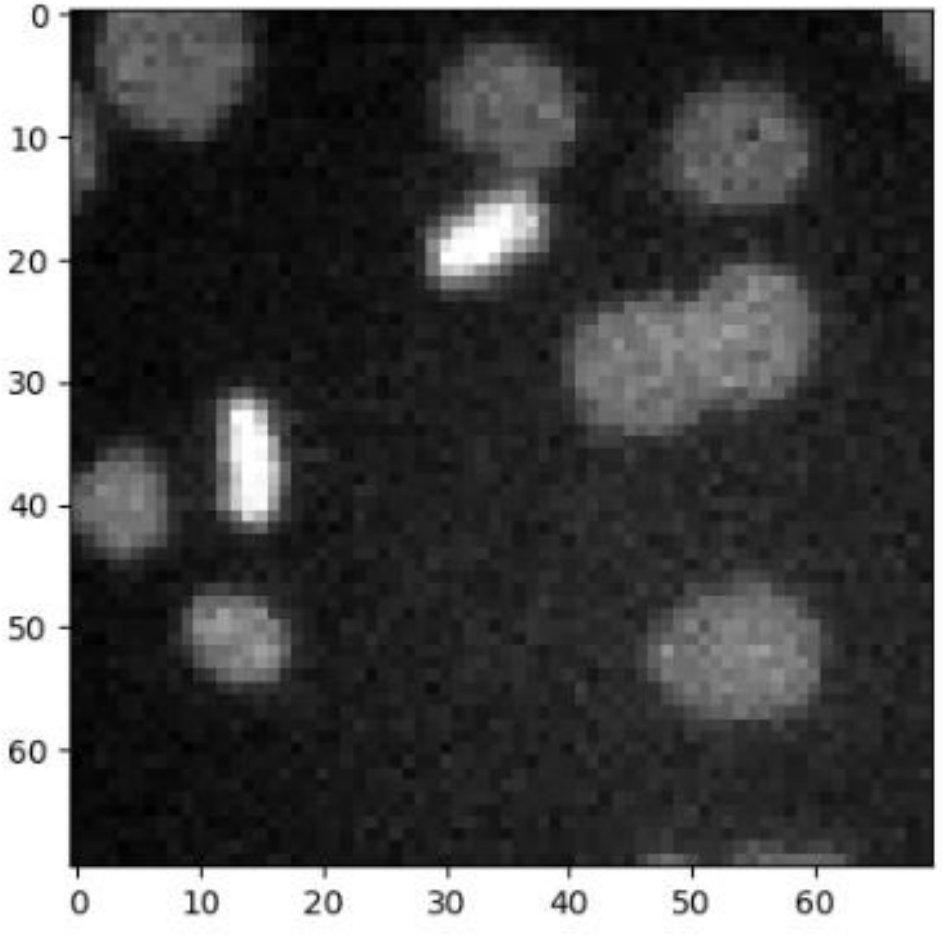
Image data source: Nasreddin Abolmaali, TU Dresden

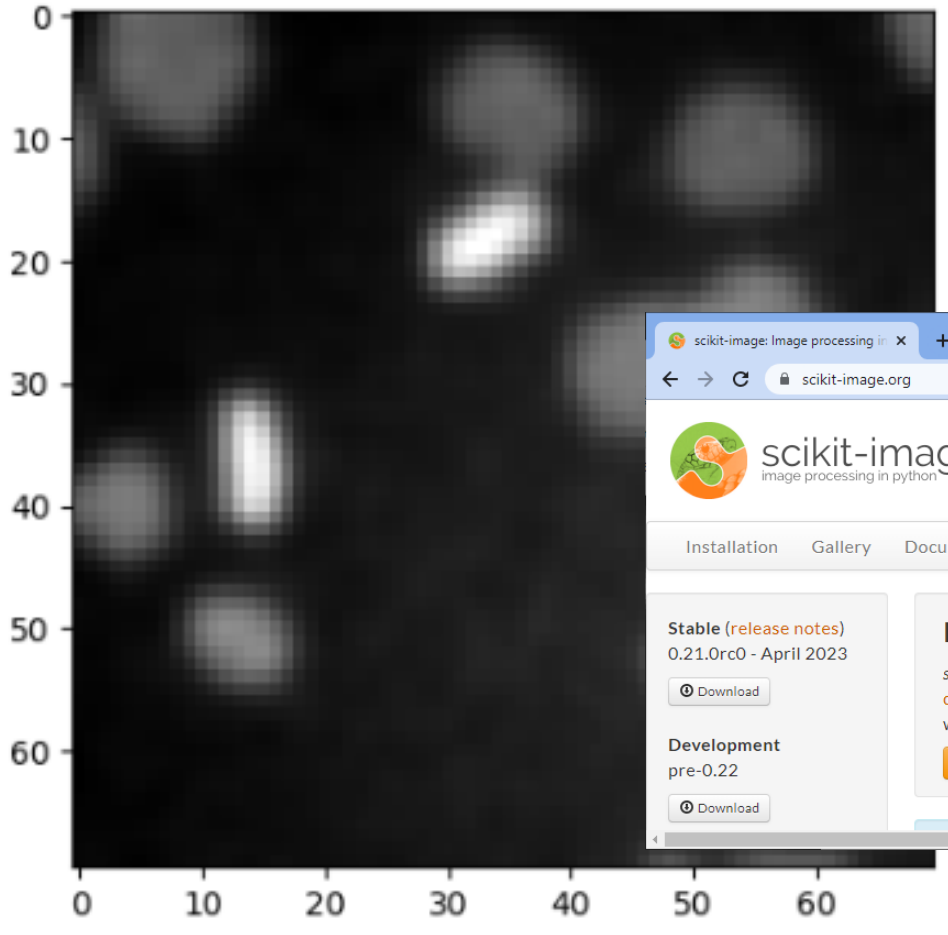
Filters

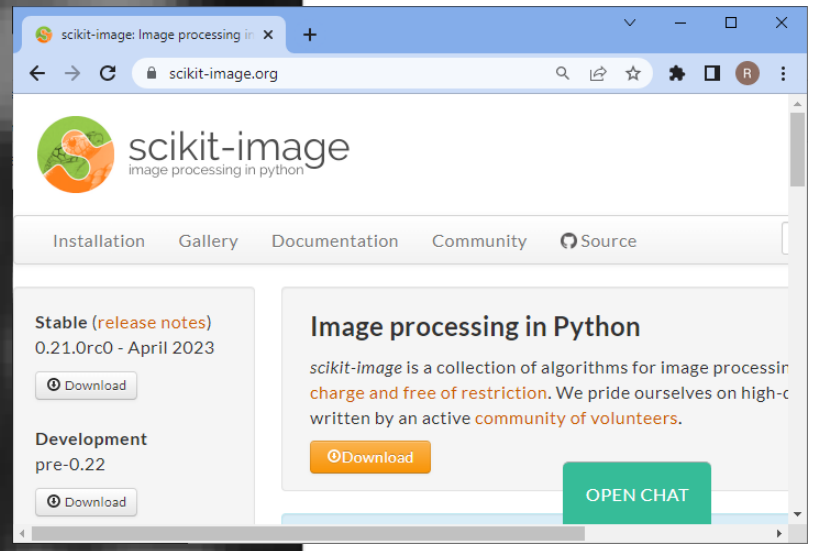
… are just functions

```
denoised_gaussian = filters.gaussian(image3, sigma=1)

plt.imshow(denoised_gaussian, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x283aab3ba90>
```







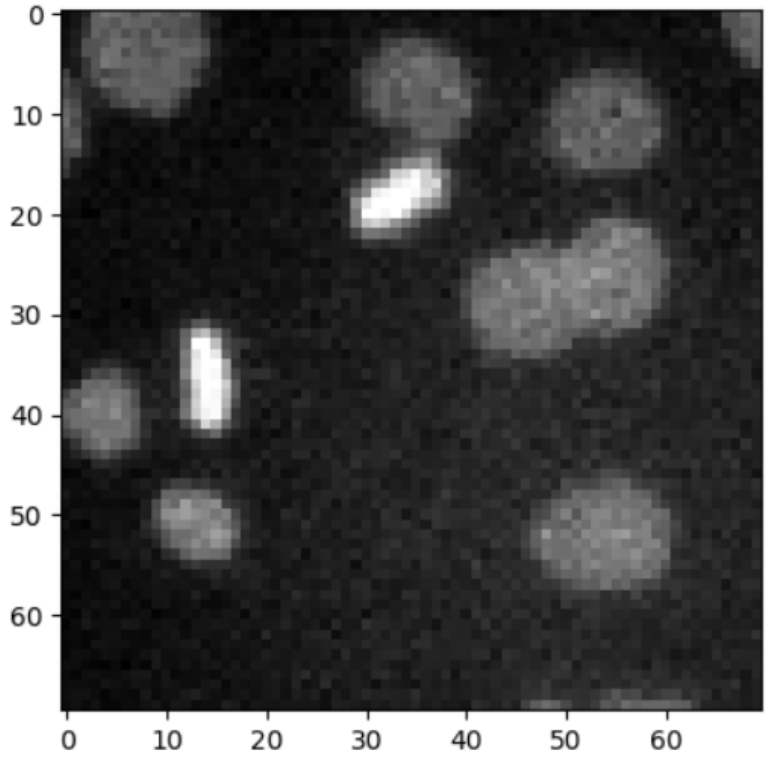
https://scikit-image.org/

Filters

- Use every opportunity and play with filter parameters to get an idea what they do.

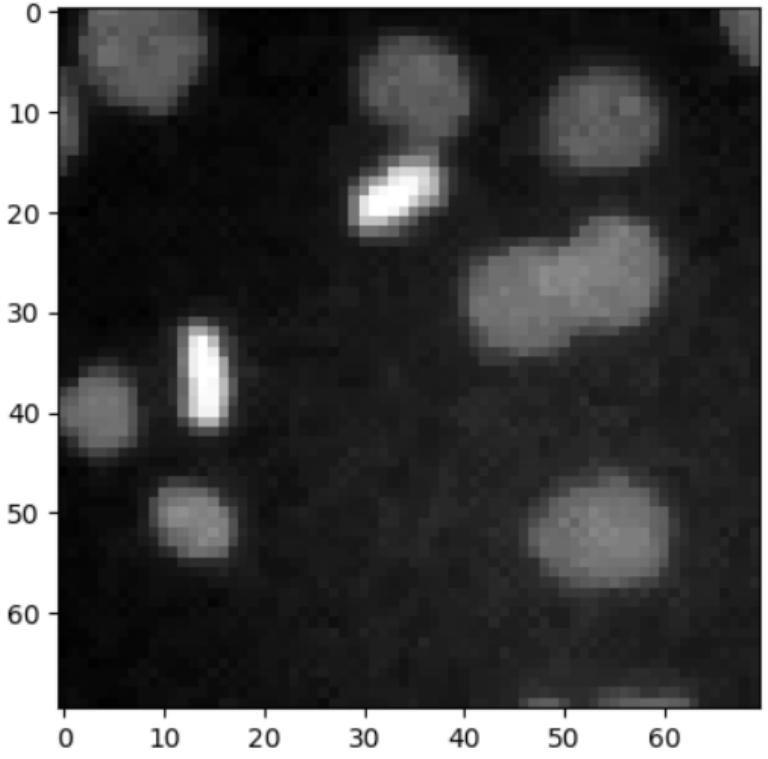
```
plt.imshow(image3, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1d86893b6d0>
```



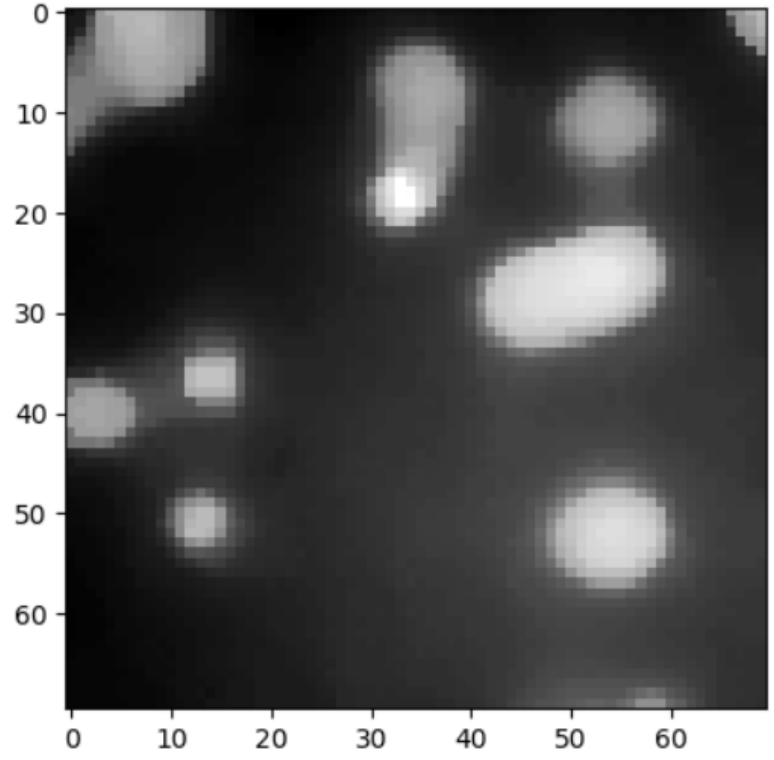

```
denoised_median = filters.median(image3, morphology.disk(1))

plt.imshow(denoised_median, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1d868a189d0>
```




```
denoised_median2 = filters.median(image3, morphology.disk(5))

plt.imshow(denoised_median2, cmap='gray')
```

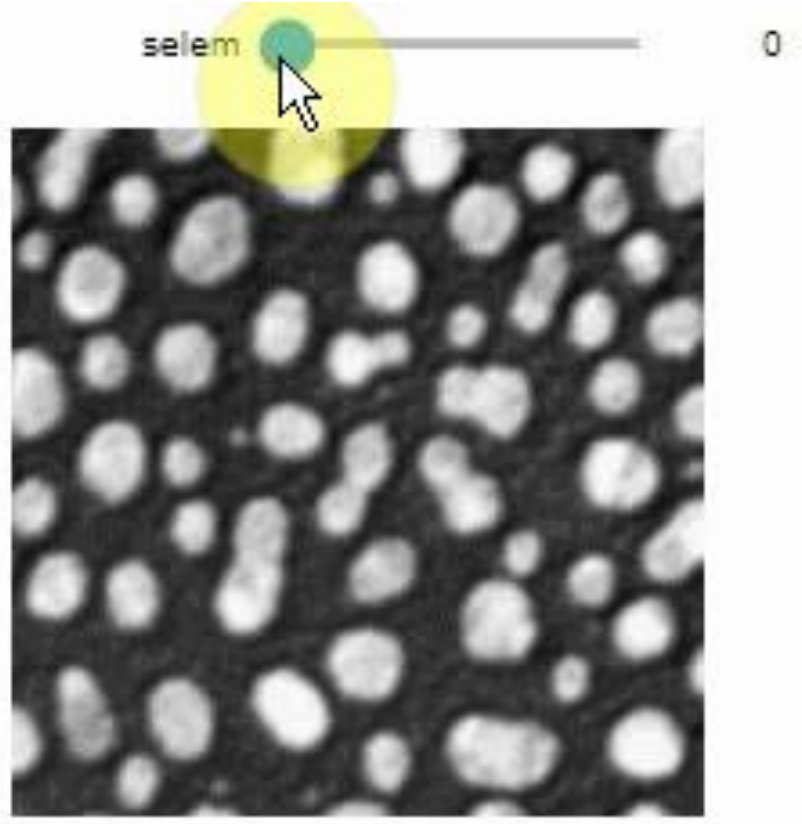
```
<matplotlib.image.AxesImage at 0x1d868ca7af0>
```



Filters

- Use every opportunity and play with filter parameters to get an idea what they do.

```
from skimage.filters.rank import maximum
stackview.interact(maximum, slice_image)
```



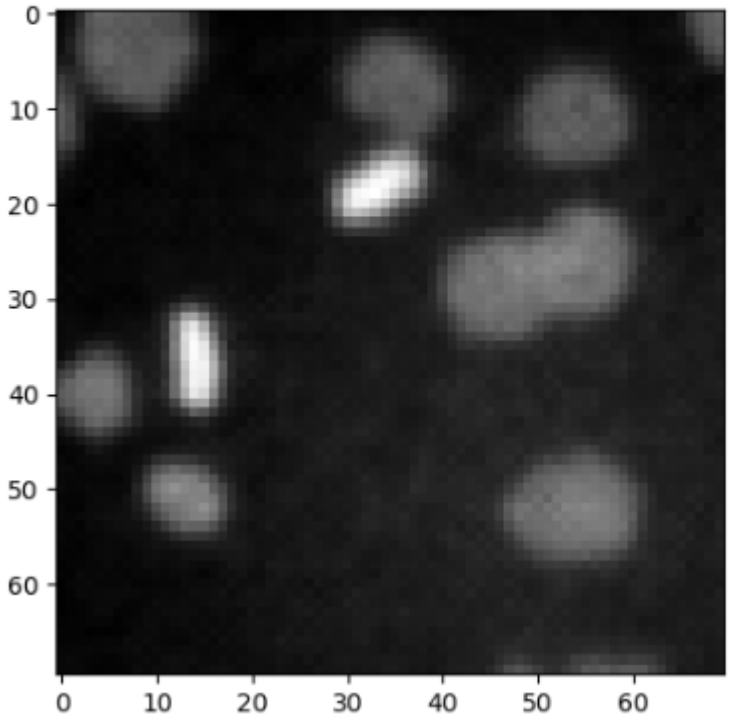
Filters

PoL Physics of Life TU Dresden

… are just functions

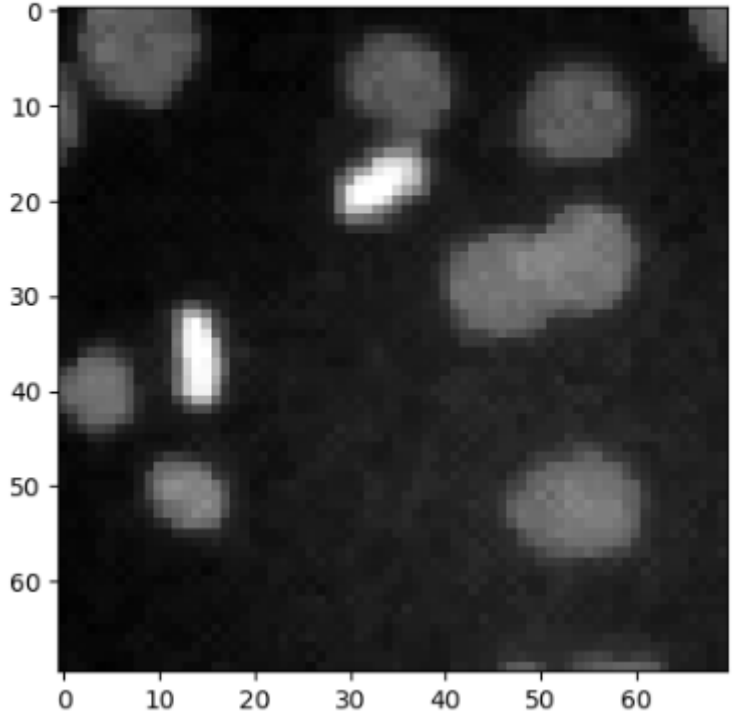
```
denoised_mean = filters.rank.mean(image3.astype(np.uint8), morphology.disk(1))

plt.imshow(denoised_mean, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x283a9868310>
```



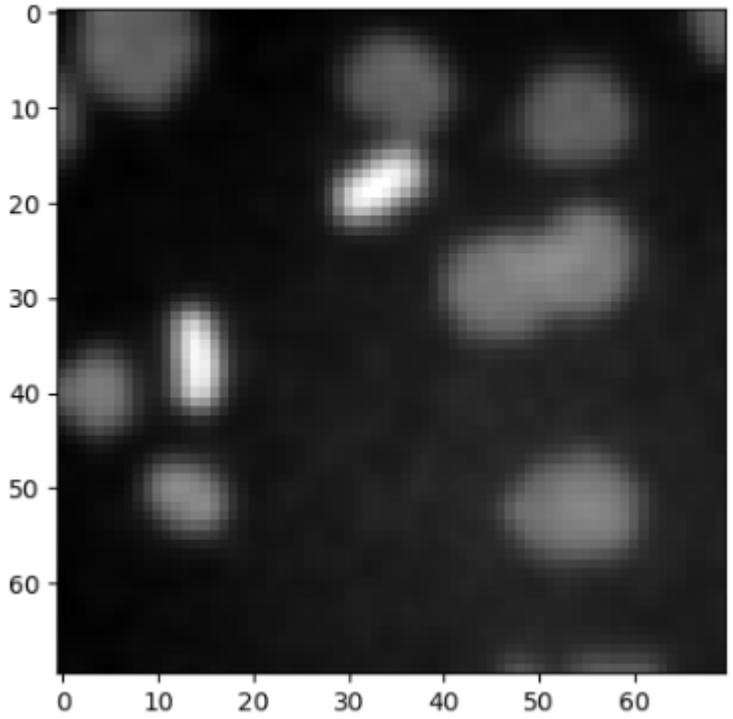

```
denoised_median = filters.median(image3, morphology.disk(1))

plt.imshow(denoised_median, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x283a98f2640>
```




```
denoised_gaussian = filters.gaussian(image3, sigma=1)

plt.imshow(denoised_gaussian, cmap='gray')
```

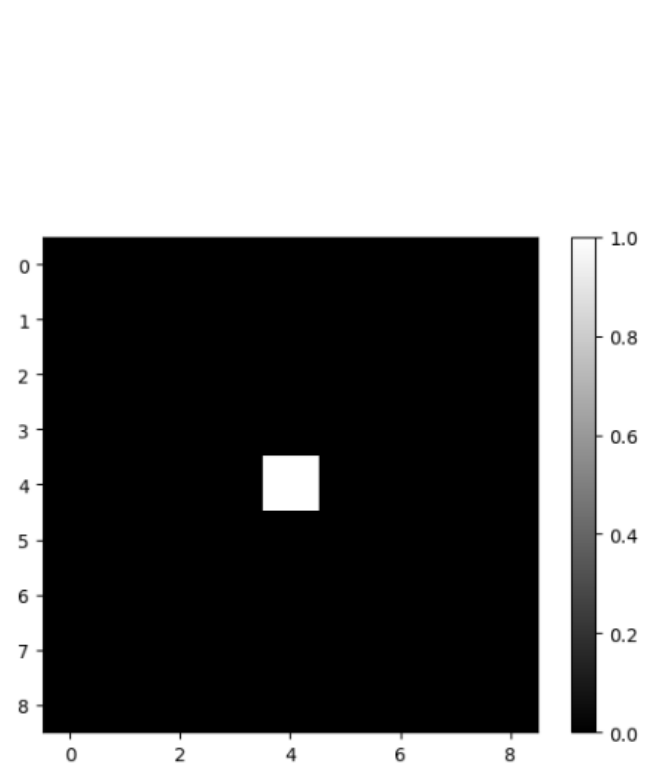
```
<matplotlib.image.AxesImage at 0x283aab3ba90>
```



Filters

... may be custom functions

Recommendation: Apply custom filters to super simple images to see if they do the right thing.



```python
def laplacian_of_gaussian(image, sigma):
    """
    Applies a Gaussian kernel to an image and the Laplacian afterwards.
    """

    # blur the image using a Gaussian kernel
    intermediate_result = filters.gaussian(image, sigma)

    # apply the mexican hat filter (Laplacian)
    result = filters.laplace(intermediate_result)

    return result
```

```python
log_image1 = laplacian_of_gaussian(image2, sigma=1)

plt.imshow(log_image1, cmap='gray')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x283a9679430>
```

Binarization / Thresholding

- Turn images into binary images (very basic form of segmentation)

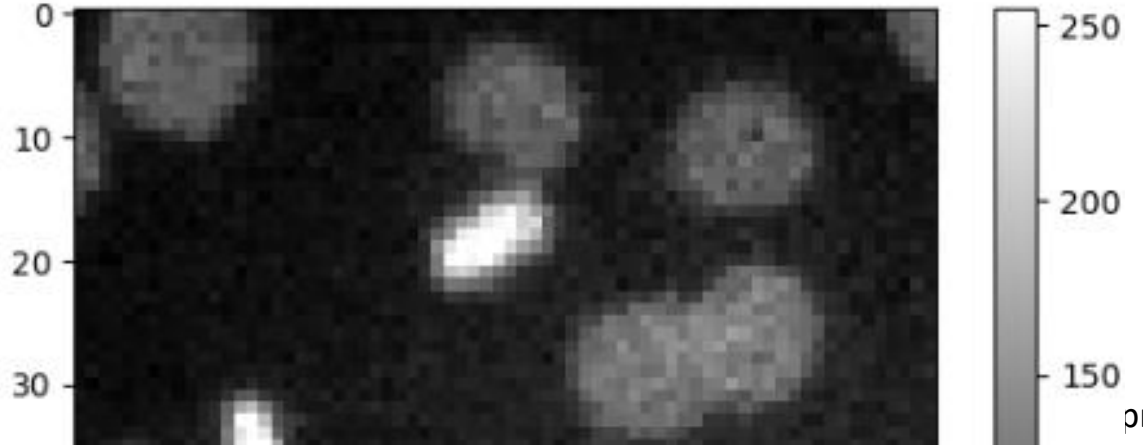
- When using scikit-image, `threshold_` functions typically return a threshold you need to apply yourself.

```
from skimage.filters import threshold_otsu

threshold = threshold_otsu(image_nuclei)
threshold
```

```
77
```

```
image_otsu_binary = image_nuclei > threshold

plt.imshow(image_otsu_binary, cmap='gray')
plt.colorbar()
```

```
<matplotlib.colorbar.Colorbar at 0x1c285b4f550>
```



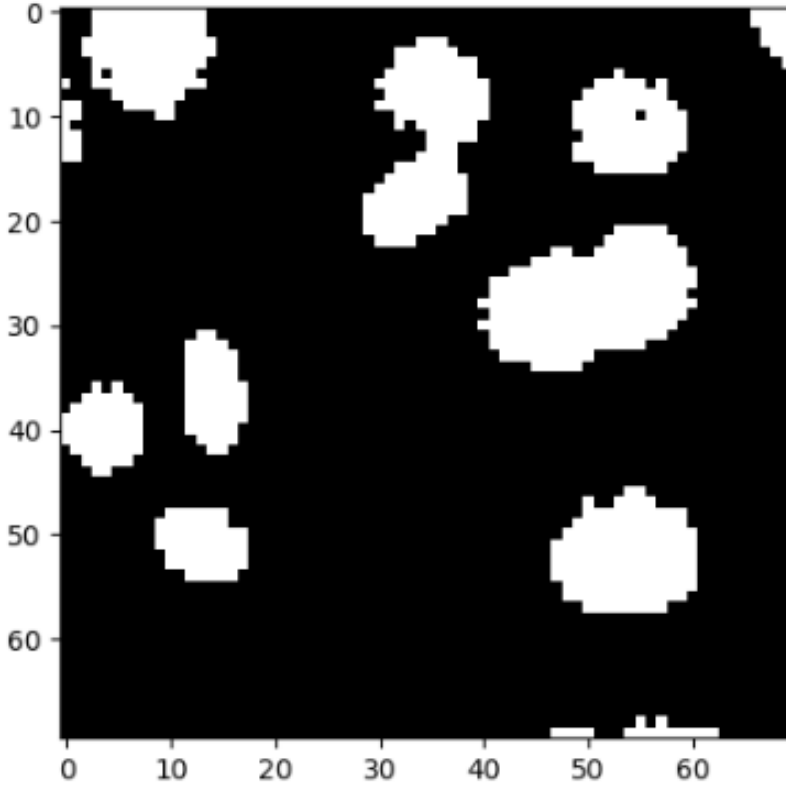
Morphological operations

- To *morph* objects in binary images

```
from skimage import morphology
```

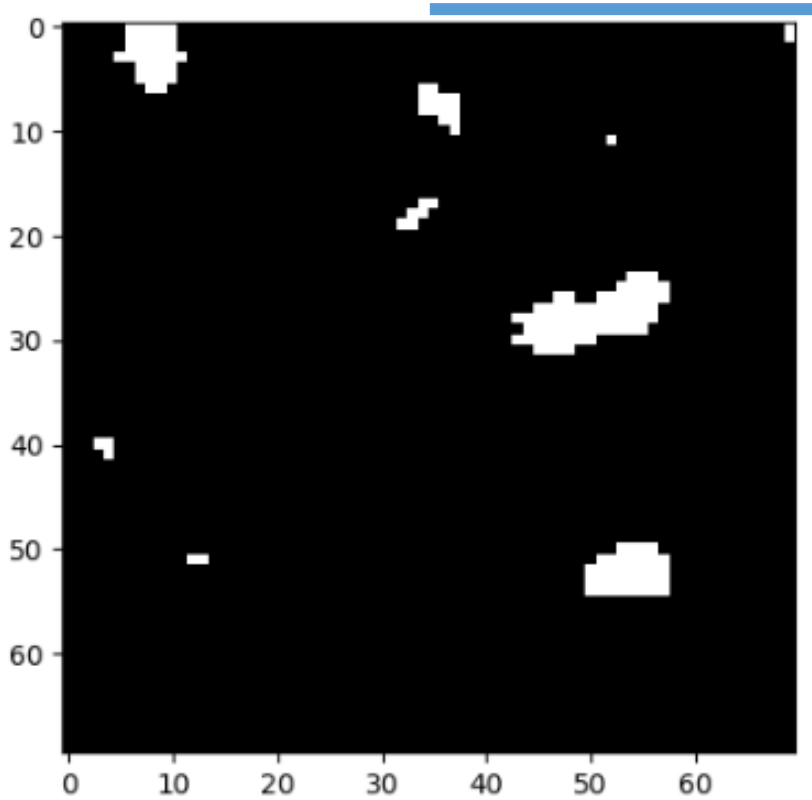
```
eroded = morphology.binary_erosion(image_binary, disk)

plt.imshow(eroded, cmap='gray')
```

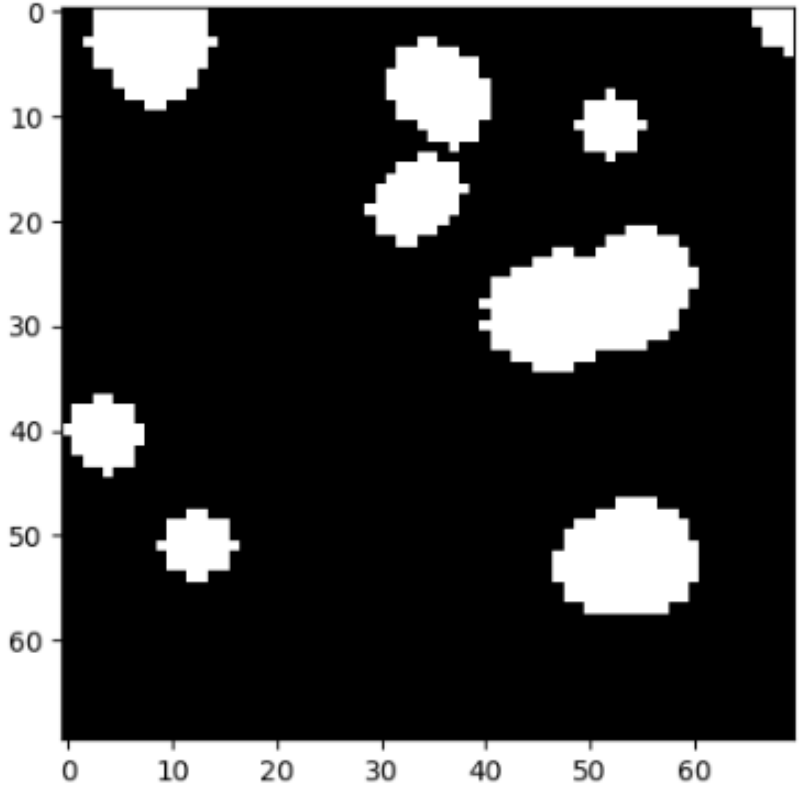
```
<matplotlib.image.AxesImage at 0x15288661520>
```




```
eroded_dilated = morphology.binary_dilation(eroded, disk)

plt.imshow(eroded_dilated, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1528893ffd0>
```





PoL Physics of Life TU Dresden

CENTER FOR SYSTEMS BIOLOGY DRESDEN

Image visualization in Python using Napari

Robert Haase

With material from

Marcelo Leomil Zoccoler, Physics of Life, TU Dresden

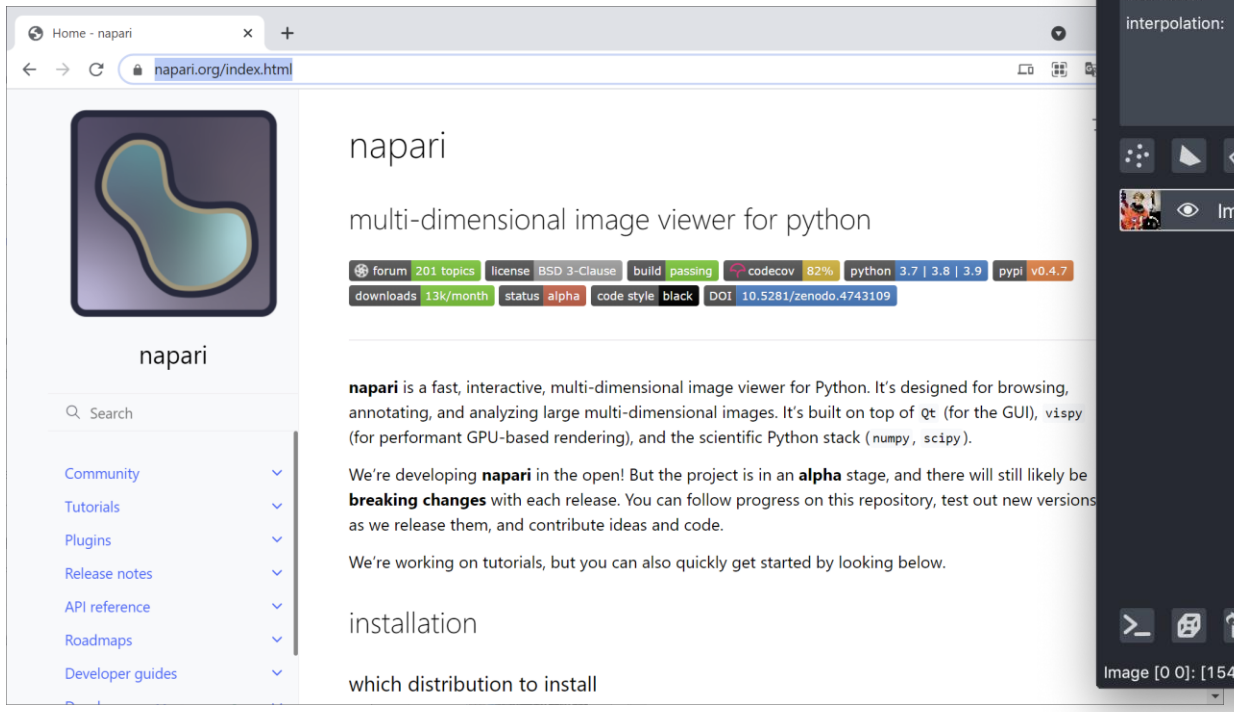
April 2023

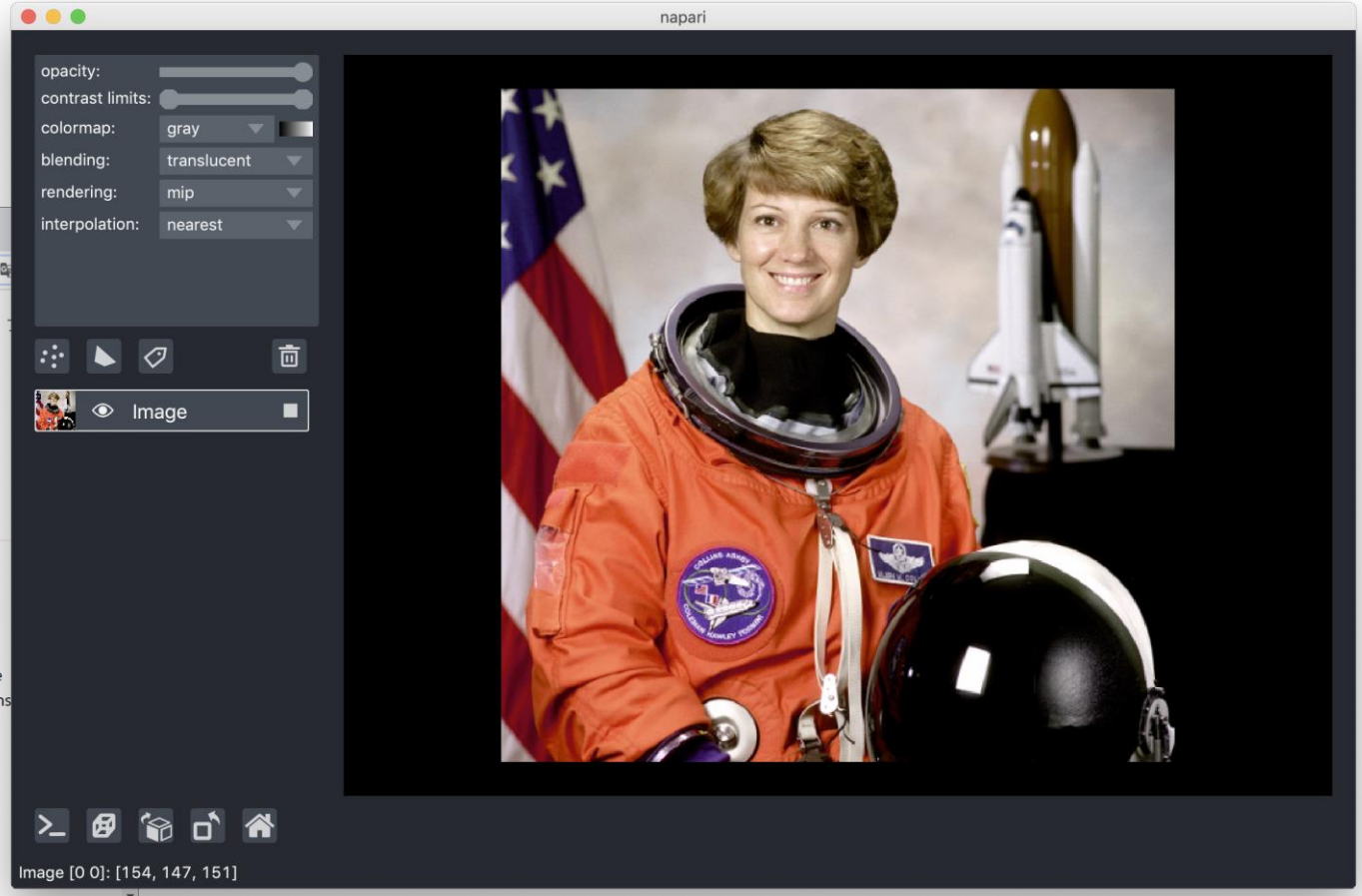
@haesleinhuepf

Napari: 3D viewer for Python

PoL Physics of Life TU Dresden

- Multi-dimensional image viewer in Python





https://napari.org/

Napari: 3D viewer for Python

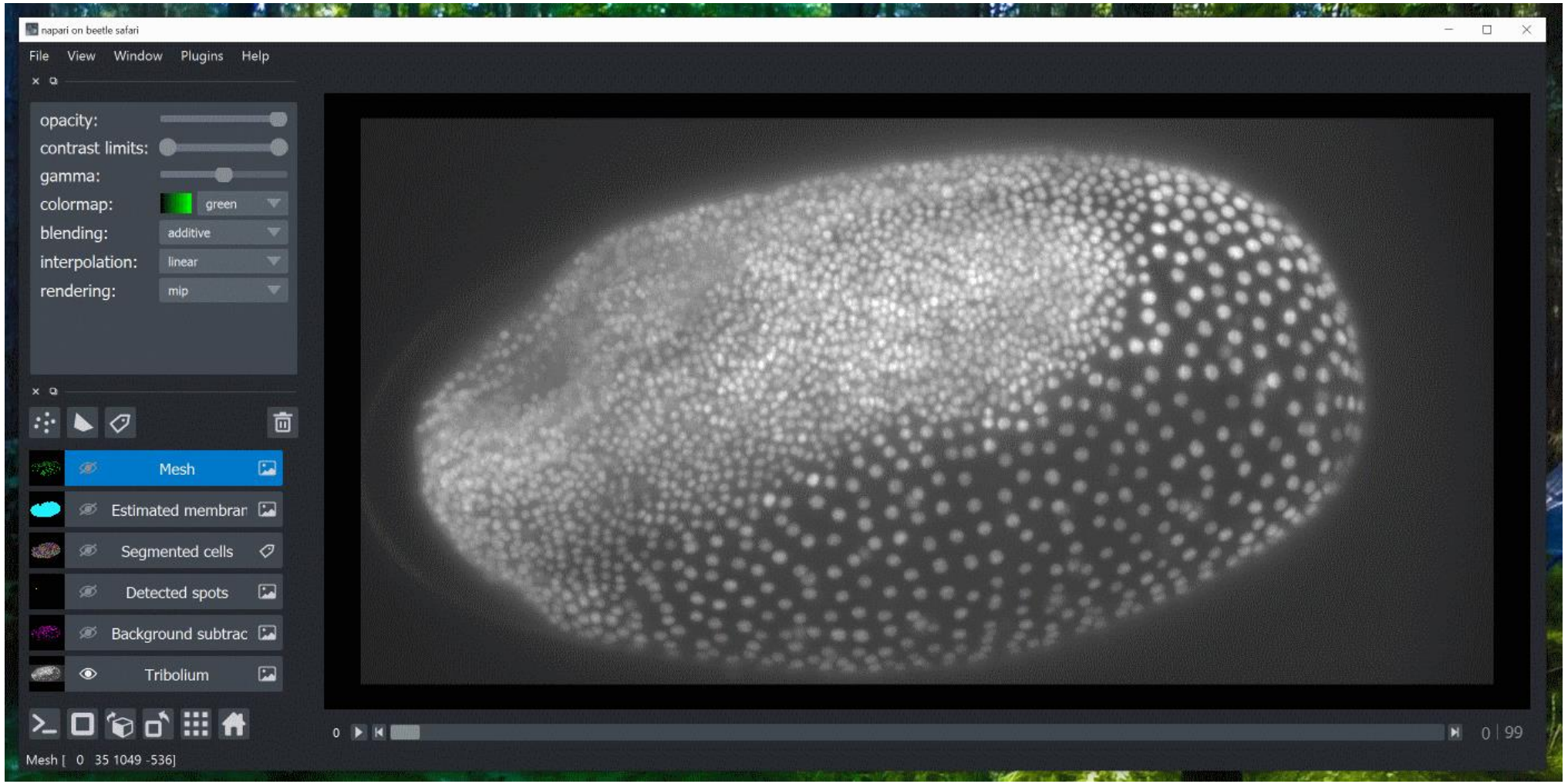


Image data source: Daniela Vorkel, Myers lab, MPI-CBG/CSBD

https://napari.org/

Napari user interface

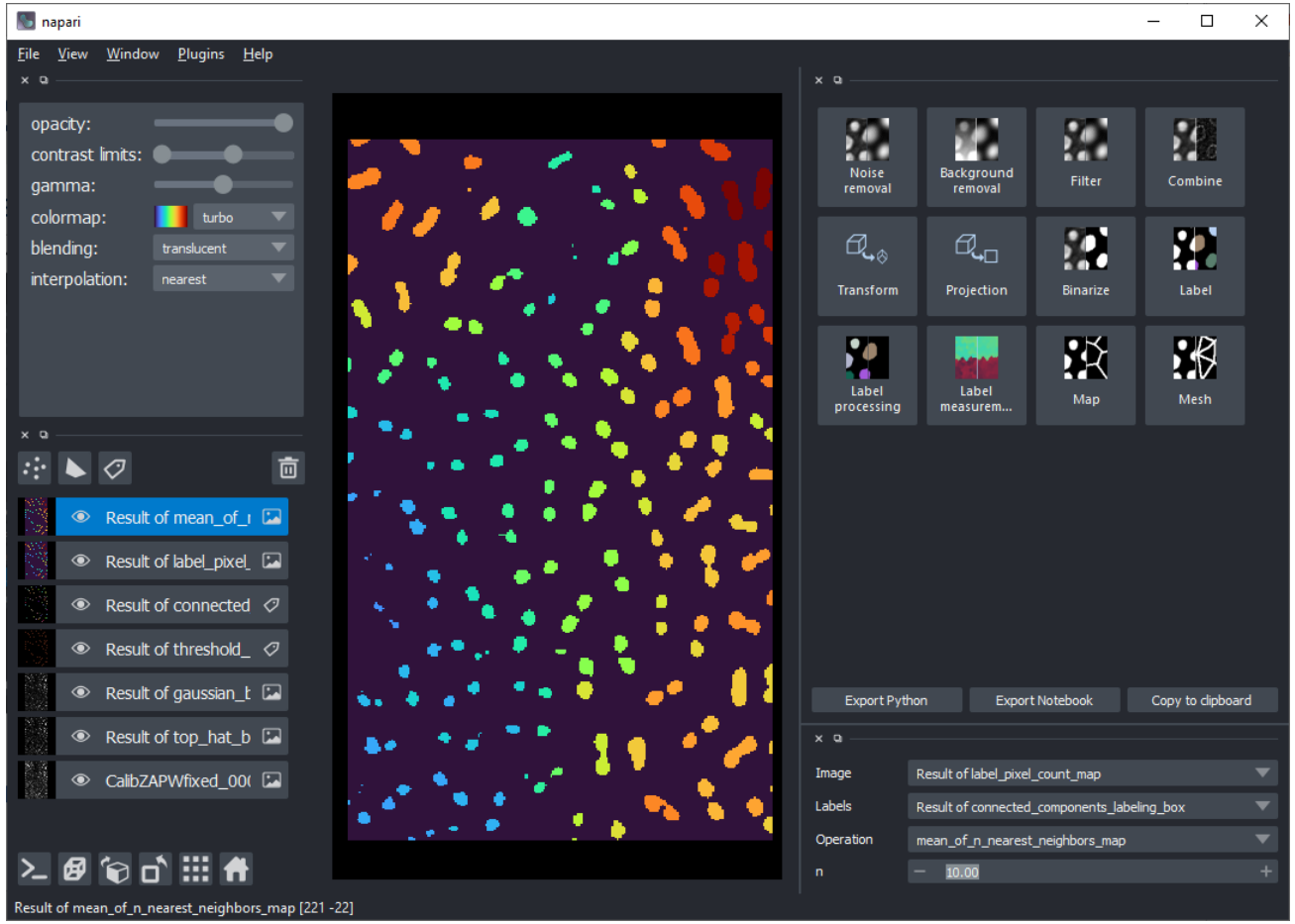
View configuration / tools

```
layer.opacity = 0.5
```

Layers

```
layer.visible = False
```

Viewer controls



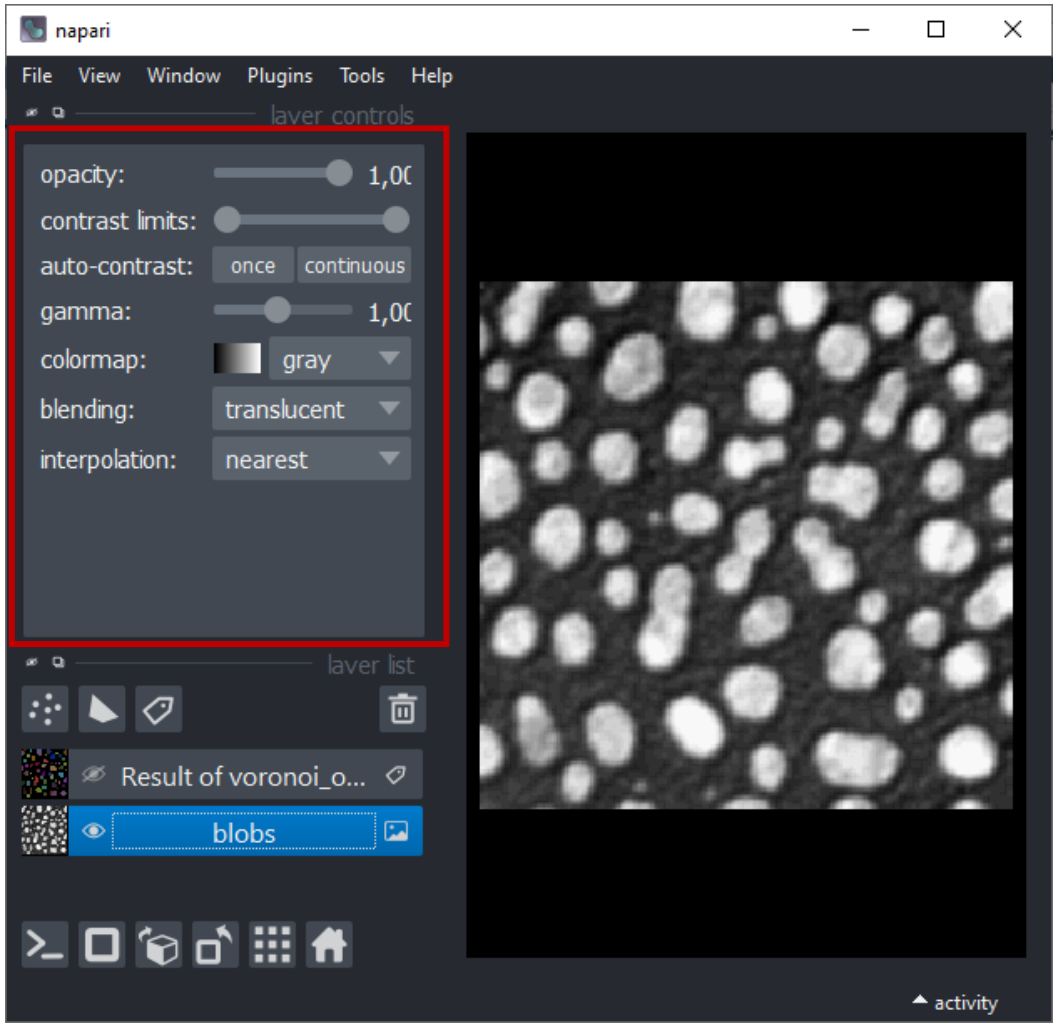
Dock widgets (custom plugins)

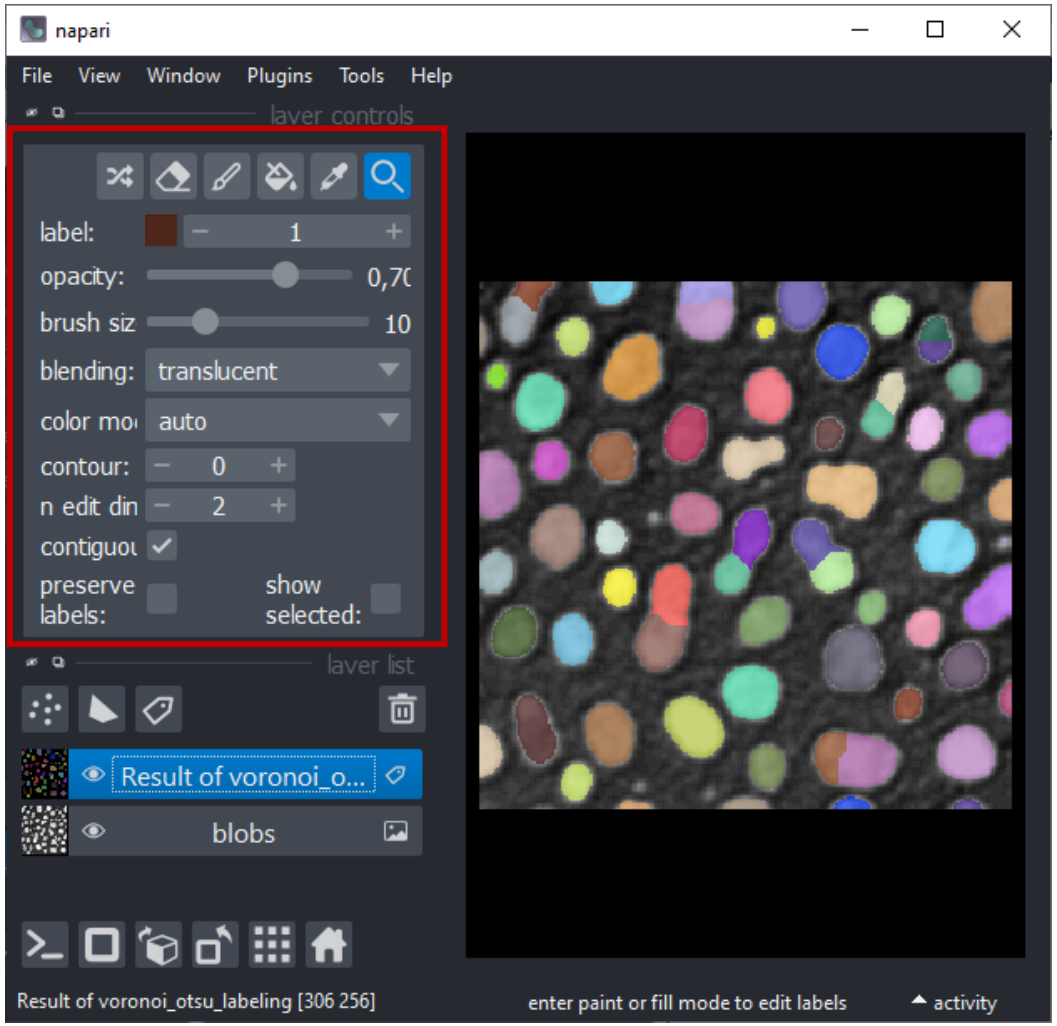
Function widgets (custom plugins)

https://napari.org/tutorials/fundamentals/viewer.html

Visualizing image segmentation

- Different layers have different configurations

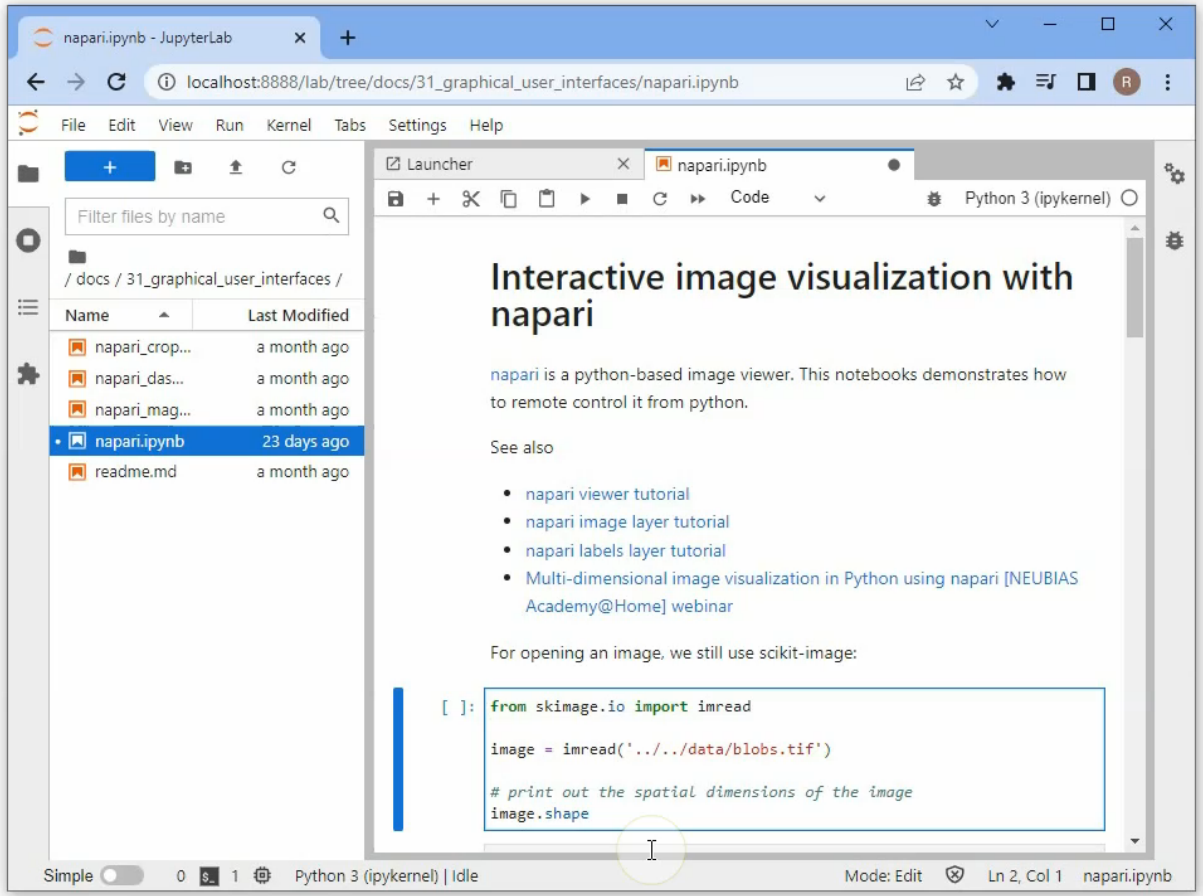




Using Napari from Python Code

PoL Physics of Life TU Dresden

- A great mix of interactivity and reproducibility



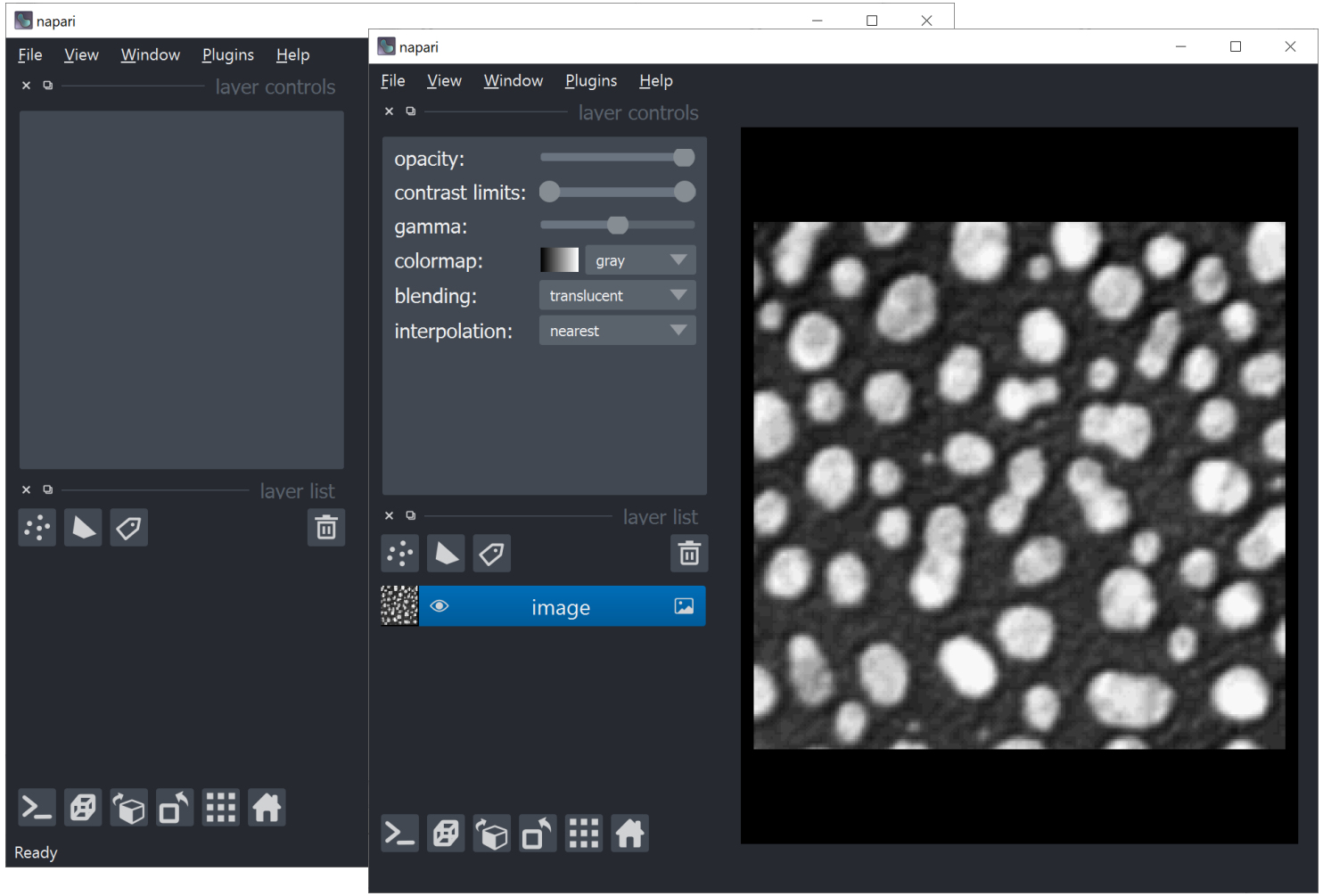
Scripting napari

- Initialization

```
import napari

# Create an empty viewer
viewer = napari.Viewer()
```

- Adding images

```
viewer.add_image(image)
```

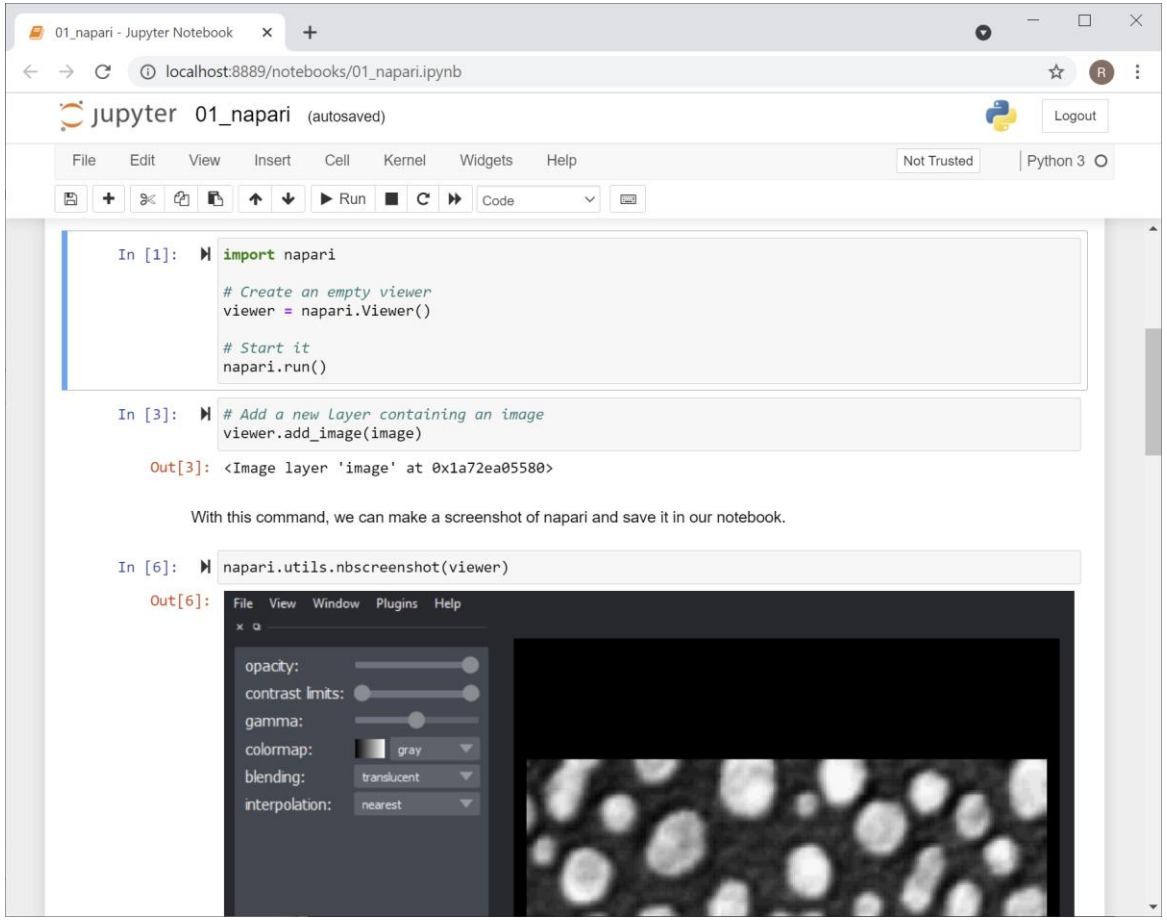


Scripting napari in notebooks

- Make screenshots from napari and put them in your jupter notebook

```
napari.utils.nbscreenshot(viewer)
```

Place your viewer here



Working with layers

- Removing layers

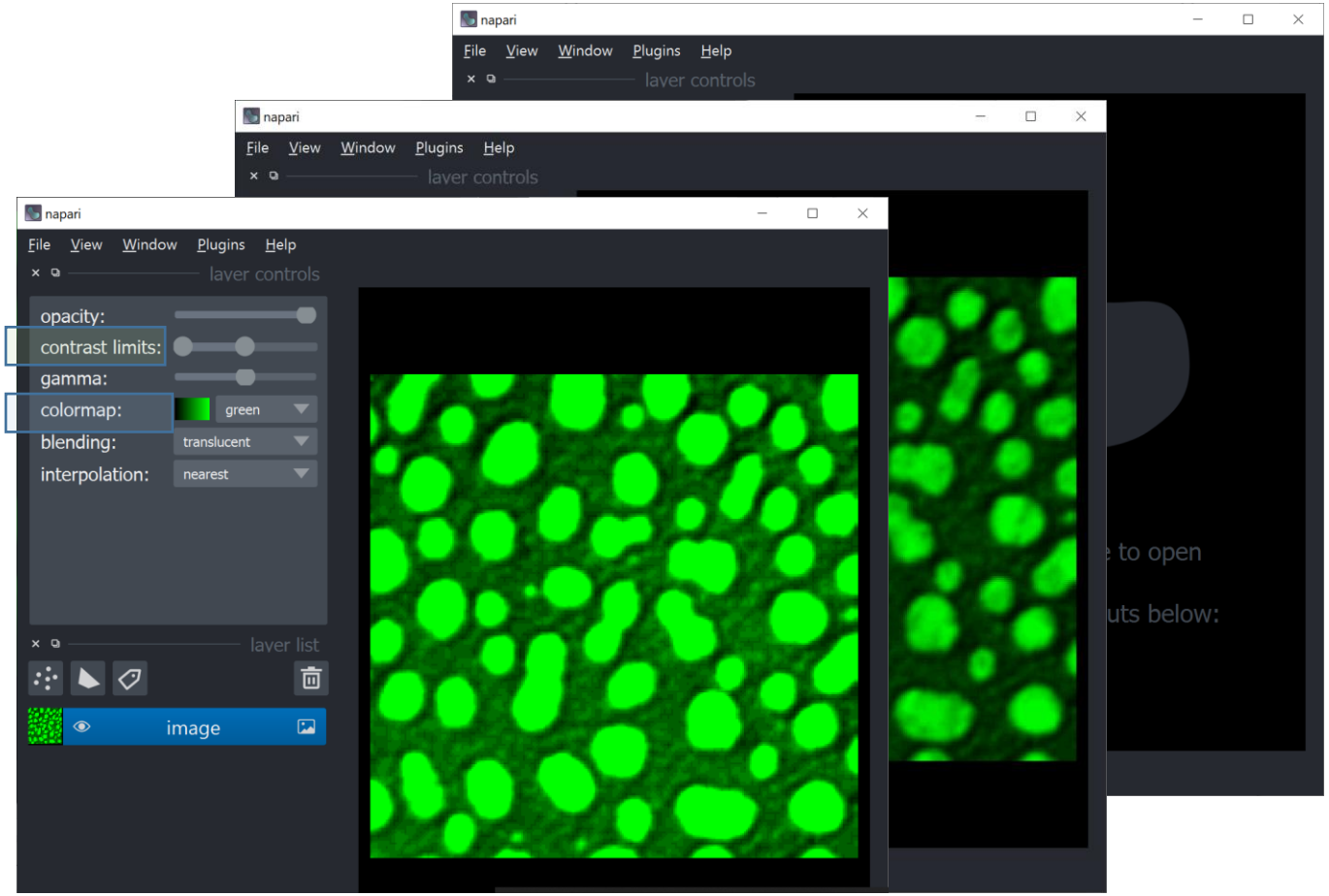
```
for l in viewer.layers:
    viewer.layers.remove(l)
```

- Modify visualization while adding layers

```
viewer.add_image(image,
                 colormap='green’)
```

- Modify layers after adding

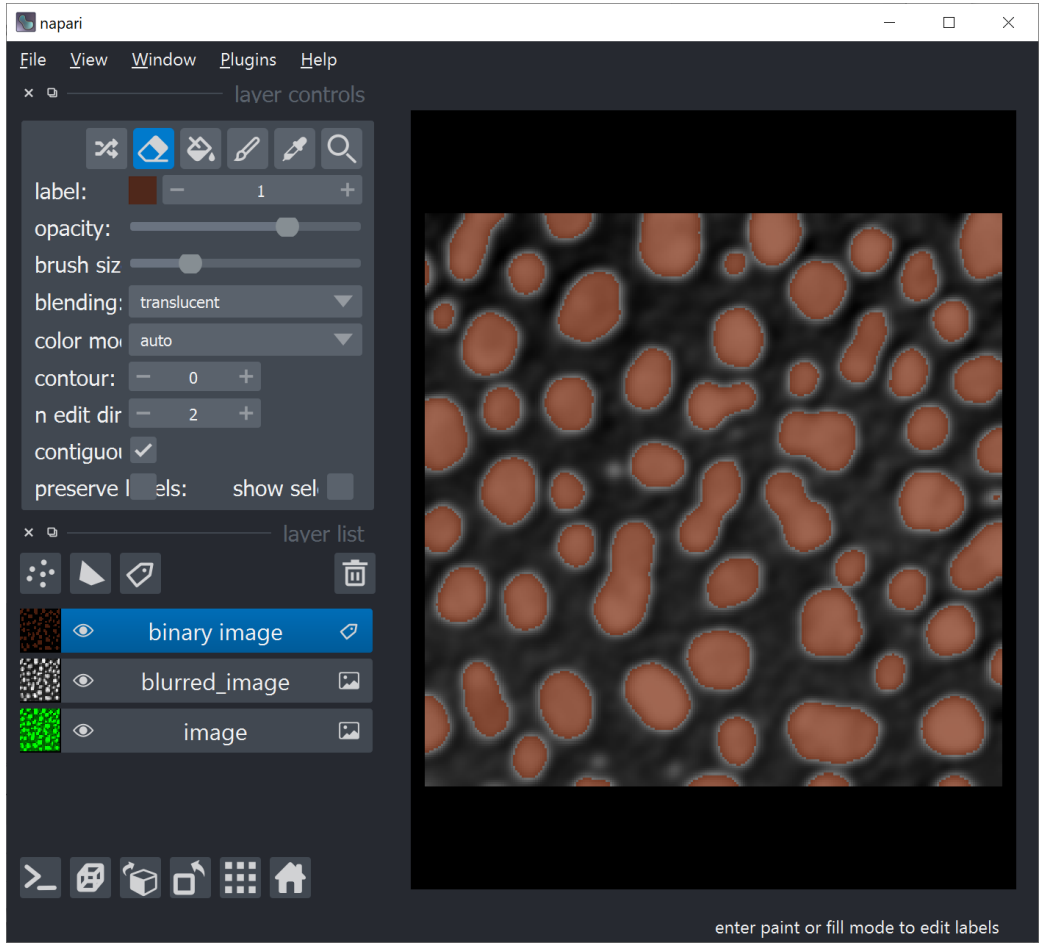
```
layer = viewer.add_image(image)
layer.colormap = 'green’
layer.contrast_limits = (0, 128)
```



Visualizing image segmentation

- Binary images and label images visualized as label layers

```
from skimage.filters import threshold_otsu
threshold = threshold_otsu(blurred_image)
binary_image = blurred_image > threshold

# Add a new labels layer containing an image
viewer.add_labels(binary_image)
```



Summary

- Image visualization
  - Pixel size, colormaps, bit-depth
  - Image histogram
  - Brightness/Contrast
- Image Filtering
- Morphological Operations
  - Mask Refinement

- Python libraries
  - Matplotlib
  - Scikit-image
  - Napari

Coming up next

- Image Segmentation
  - Connected component analysis
  - Voronoi-Otsu-Labeling
- Surface reconstruction